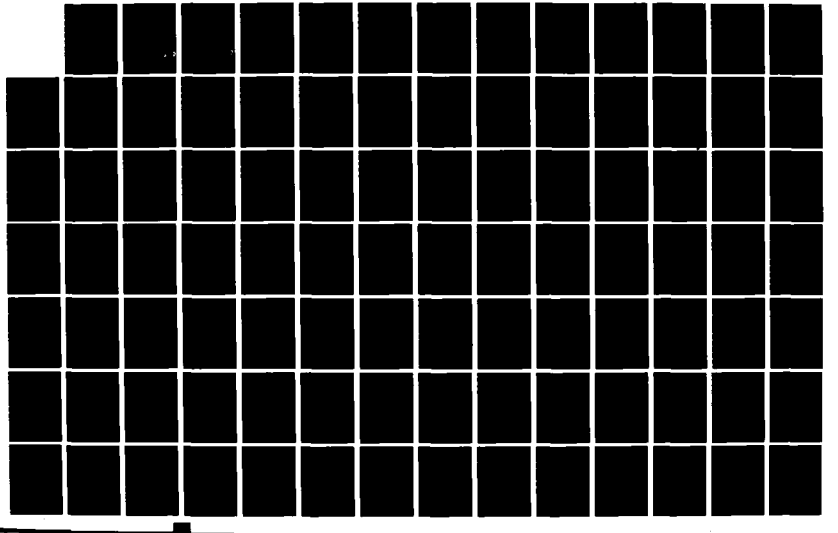
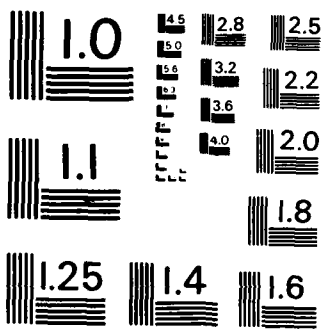
AD-A161 715 AN ASSESSMENT OF ADA'S SUITABILITY IN GENERAL PURPOSE PROGRAMMING APPLICATIONS(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF SYST.. 1/3

UNCLASSIFIED L D CAVITT ET AL. SEP 85 F/G 9/2 NL





(2)

AD-A161 715

AN ASSESSMENT OF ADA'S SUITABILITY IN GENERAL PURPOSE PROGRAMMING APPLICATIONS

THESIS

Larry D. Cavitt
Captain, USAF

Anthony A. Panek
Captain, USAF

AFIT/GLM/LSM/85S-62

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GLM/LSM/85S-62

AN ASSESSMENT OF ADA'S SUITABILITY IN GENERAL PURPOSE PROGRAMMING APPLICATIONS

THESIS

Larry D. Cavitt
Captain, USAF

Anthony A. Panek
Captain, USAF

AFIT/GLM/LSM/85S-62

Approved for public release; distribution unlimited

The contents of the document are technically accurate, and no sensitive items, detrimental ideas, or deleterious information are contained therein. Furthermore, the views expressed in the document are those of the author(s) and do not necessarily reflect the views of the School of Systems and Logistics, the Air University, the United States Air Force, or the Department of Defense.

AFIT/GLM/LSM/85S-62

AN ASSESSMENT OF ADA'S SUITABILITY IN GENERAL PURPOSE PROGRAMMING APPLICATIONS

THESIS

Presented to the Faculty of the School
of Systems and Logistics
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Logistics Management

Larry D. Cavitt
Captain, USAF

Anthony A. Panek
Captain, USAF

September 1985

Acknowledgements

While the collective assistance of many individuals at the Air Force Institute of Technology contributed to this reseach, the authors wish to specifically acknowledge those who made major contributions.

We would like to thank our advisor Captain Patricia Lawlis for her guidance and assistance during this research effort, who encouraged and did not let us stray from our research objectives.

We would also like to thank Dr. Charles Richard for the assistance he gave us while learning the Ada language.

Finally, we wish to thank our wives for their continuing patience, encouragement and understanding (except during late Friday-night sessions at the Fly-Wright) which inspired us to complete this research project on time.

Table of Contents

Page

List of Figures

List of Acronyms

AFIT	Air Force Institute of Technology
AJPO	Ada Joint Program Office
APSE	Ada Programming Support Environment
DARPA	Defense Advanced Research Project Agency
DCA	Defense Communications Agency
DPD	Data Project Directive
HOL	High Order Language
HOLWG	High Order Language Working Group
KAPSE	Kernal Ada Programming Support Environment
LRM	Language Reference Manual
NSA	National Security Agency
PDL	Program Design Language

Abstract

The Ada programming language is the result of a multiyear effort under the sponsorship of the Department of Defense (DoD) to obtain the benefits of a single DoD-wide language for use in embedded computer systems. The language was developed to reduce or eliminate many of the serious and costly problems associated with the development and maintenance of software for embedded systems. This research assesses Ada's suitability in simple, non-embedded applications, specifically, numerical computation, simulation, and file processing. FORTRAN and Pascal programs in these applications were translated into Ada. Comparisons were made between the originals and the translations with regard to lines of source code, transportability, maintainability, readability, execution time, and any other finding relevant to the study. The study revealed that while further research is needed, Ada is a powerful programming language suitable for use in these non-embedded applications.

AN ASSESSMENT OF ADA'S SUITABILITY IN GENERAL PURPOSE PROGRAMMING APPLICATIONS

I. OVERVIEW

Introduction

In the early 1970's the Department of Defense (DoD) conducted studies on the proliferation of computer programming languages in DoD. On the basis of the studies it was predicted that $24 billion could be saved on DoD computer software costs between 1983 and 1999 if one common programming language was used rather than the approximately 450 programming languages and incompatible dialects then in use in DoD. The area of computer application with the greatest number of different languages and military service unique versions of languages involved embedded computer systems, and hence this area was chosen as the original target application area for a new common DoD language. An embedded computer is one which is an integral part of a larger system and either controls or otherwise affects the operation of the system. Embedded computers are part of virtually every military weapon system today (2:12-13). DoD, in an effort to standardize and replace most of the programming languag-

es in use, sponsored the development of the Ada programming language. Although Ada was developed primarily for use in embedded systems, it also has the potential to be used in the general purpose programming environment. That is, Ada has the potential to be used in a wide range of applications, such as payroll, inventory management, numerical computation, and personnel data. If Ada can effectively be used as a general purpose language and becomes the DoD standard language for all applications, as opposed to just embedded systems, costs associated with support of all programming languages used for non-embedded applications can also be eliminated. This will result in cost savings of more than the original estimate of $24 billion (2:12-13; 3:31; 13:9).

Problem Statement

Eventually, DoD will require all embedded systems to be written in Ada. Therefore, current research efforts are primarily aimed at evaluating embedded applications. This study is not involved with embedded systems, but rather investigates the suitability of Ada in other than embedded system applications. In particular, this study analyzes Ada against other traditional languages as they are used in particular applicationns. The languages and applications evaluated are; 1) FORTRAN, in a numerical computation and in a simulation application, and 2) Pascal, in a text file

processing application. This study evaluates the relative advantages and disadvantages of using Ada versus the chosen language in the given application.

Justification

The Department of the Air Force, Directorate of Information and Technology, issued Data Project Directive (DPD) HAF-P83-Ø06, dated 23 December 1983. The DPD "directs planning, experimentation, and analysis efforts required to evaluate the use of Ada in the general purpose computing environment. This program is in pursuit of Ada Joint Program Office efforts to implement and introduce Ada within DoD as provided by their charter, (OUSD(R&E)) memo, 12 December 198Ø, Ada Joint Program Office (AJPO)" (19:1).

The objective of the program is to evaluate Ada in a general purpose environment and to identify training requirements. Specifically, the DPD directs the participants in the study to:

> 1. Gain experience with Ada by using the language to accomplish a representative range of end uses.
>
> 2. Document experiences using Ada and provide a technical evaluation of the suitability of the language for widespread use in the Air Force general purpose computing arena or subsets thereof (19:1).

The DPD has specifically tasked the Air Force Institute of Technology to "use Ada on one or more selected applications and provide evaluation reports in pursuit of the stated objectives" (19:3).

The justification for this research is based on the DPD. Research using Ada in the general purpose environment is necessary as it will be used as part of the basis in the determination of the suitability of Ada as the DoD standard language.

Scope

The scope of this research is confined to the evaluation of the Ada programming language in general purpose applications. As mentioned earlier, this study is limited to an evaluation of Ada versus the high order programming languages FORTRAN and Pascal. The applications considered are numerical computation and a simulation application in FORTRAN, and a text file processing application using Pascal.

Limitations

Two of the compilers used in this study are the TeleSoft-Ada Compiler, version 1.5, 31 May 1983, and the TeleSoft-Ada Compiler, version 2.2, 11 Feb 1985, running under the UNIX operating system.

Both versions of the TeleSoft-Ada Compilers are unvalidated by the Ada Joint Program Office and are only a partial implementation of the full Ada language. Although only a subset of the full language is available, the unvalidated compilers implement enough of the language facilities to be useful in this research. All language facilities

available in this release of the compiler conform to the requirements of ANSI/MIL-STD 1815A, 22 Jan 1983, and will therefore be in subsequent releases, and ultimately, the validated version.

This research does not involve an evaluation of Ada against COBOL applications. Compete assessment of Ada's suitability as a general purpose language must include comparisons with a language so widely used for business data processing as COBOL. However, the authors have no previous experience with COBOL, therefore, they did not attempt to evaluate the differences.

Research Objectives

The objectives of this research effort are as follows:

1. To determine if Ada is suitable as the implementation programming language in the applications chosen for the study.

2. Identify particular strengths and/or weaknesses of Ada in the specific applications and in general.

3. Make recommendations from the findings on the suitability of Ada as a general purpose programming language.

Research Questions

This study addresses Ada's strengths and weaknesses relative to the language it is to be compared against, and includes but is not limited to the following areas:

1. To what extent can the Ada translated programs replicate the output of the original programs?

2. What differences, if any, are evident concerning the number of lines of source code and the size of the executable code necessary to replicate the output as compared against the original programs?

3. Are there differences in the runtime characteristics of the programs coded in Ada as opposed to the original programs?

4. Are there any differences in the maintainability and transportability aspects of the Ada coded programs to include error detection, testability, and any other observations relevant to the maintenance of the complete system as compared against the original programs?

5. What are the differences or similarities in readability of the source codes?

6. Other findings which are important to the overall evaluation of the language in the general purpose environment.

II. LITERATURE REVIEW

Introduction

This literature review investigates several aspects of the computer programming language Ada. The literature review will first identify problems which led to the need for a common high order language; second, provide an overview of the steps taken to develop Ada; third, relate Ada's features to modern programming methodologies; and finally, outline what the literature reveals concerning Ada's suitability to be used as a general purpose programming language.

Background: General Issue

In the early 1970's, the United States Department of Defense (DoD) was faced with an increasing trend in software costs. In 1973, software costs were over $3 billion, and consisted of 46 percent of DoD computer costs. A breakout of these costs by computer application reveals: 56 percent for embedded systems, 19 percent for data processing, 5 percent for scientific, and 20 percent for indirect software costs. An early 1970's study by the Electronics Industries Association (EIA) predicted that total DoD software costs for embedded systems alone would exceed $32 billion in 1990. Software shortcomings within DoD which created these rising

costs were a diversity of programming languages, improper application of programming languages, languages not equipped to handle modern programming methodologies, and a lack of useful software environments. An early 1970's DoD study revealed over 450 different programming languages in use within DoD, resulting from the lack of controls on the use of computer languages. Project managers were free to use any language. All of this led to increased software costs in the following ways:

> 1. Duplication of training and maintenance for each independent language, compilers and software support packages.
>
> 2. Limiting the applicability of new support software to one system or project (11:26).

Besides cost, though related to cost, another reason for the development of a more powerful language is a condition called the "software crisis." Grady Booch describes the crisis in the following way:

> Our computers make some things more efficient and have opened areas of application that were previously impossible to solve. Correspondingly, we have developed software tools such as programming languages to help us solve problems and control our machines, but many of these tools still do not help us cope with the complexity of our solutions. Thus, software development is no longer a labor-saving activity but is labor intensive instead (2:2).

To solve this problem of complicated, unreliable, inflexible, and unmaintainable software, emphasis must be placed on developing languages which can exploit modern

design methodologies. Languages such as FORTRAN and COBOL, although popular, do not have the capabilities needed for use with modern design methodologies. According to Booch, "In a sense, these languages constrain our way of thinking about a problem to a manner that is primarily sequential and imperative; we call this condition the von Neumann mind-set" (2:3). FORTRAN and COBOL were not designed to handle the more complicated systems we currently possess, for example, embedded computer systems. Therefore, DoD needs a language which utilizes modern design techniques. David Fisher, as quoted in Booch's book, explains which software problems need to be solved:

> 1. Responsiveness. Computer-based systems often do not meet user needs.
>
> 2. Reliability. Software often fails.
>
> 3. Cost. Software costs are seldom predictable and are often perceived as excessive.
>
> 4. Modifiability. Software maintenance is complex, costly and error prone.
>
> 5. Timeliness. Software is often late and frequently delivered with less than promising capability.
>
> 6. Transportability. Software from one system is seldom used in another, even when similar functions are required.
>
> 7. Efficiency. Software development efforts do not make optimal use of the resources involved (processing time and memory space) (2:6-7).

For DoD to reduce these types of problems, a language which can support modern design methodologies was desirable.

Since the effort was aimed at embedded systems, the new language should deal with the following:

1. Parallel Processing and Real-Time Control. Capability to execute separate entities in parallel as if each were being executed by an independent logical processor. Entities proceed independently, except at rendezvous points.

2. Exception Handling. Capability of the program to respond to events that cause suspension of normal program execution because of errors or other unusual circumstances.

3. Unique I/O Control. Capability for communication with unique input and output devices.

4. Abstraction. One's view of an entity in the problem space as opposed to the view from the solution space of the computer. Part of a ladder of abstraction in which a given part of the solution is implemented at a lower level.

5. Information Hiding. To make inaccesible certain implementation details that should not affect other parts of a system (2:13, 27-28; 7:9.1).

To reduce software costs and attempt to solve the software crisis, the DoD realized the need for a common high order language. Since embedded systems comprise the majority of DoD software applicationns, the effort progressed with embedded systems in mind. The following section outlines the development of the common high order language which eventually became known as Ada (2:11-13).

Ada Development

In 1975, DoD established the High Order Language Working Group (HOLWG) to investigate the feasibility of developing a common high order language for utilization on all embedded computer systems. Membership in the HOLWG consisted of representatives from the Army, Navy, Air Force, Defense Communications Agency (DCA), National Security Agency (NSA), and the Defense Advance Research Projects Agency (DARPA). The objective of the HOLWG was to define the technical requirements for a common language, compare the requirements against existing languages and make recommendations on the adoption of a common language from existing languages or the development of a new language (12:27; 6:45).

Strawman. The first iteration of the language requirements was called Strawman. In April 1975 Strawman was distributed to the military services and other federal agencies for review. There were no quantifiable features in the Strawman. The general goals of Strawman were to determine efficiency, reliability, readability, simplicity and implementation. The reviews and responses from the Strawman document led to a tentative set of requirements called Woodenman.

Woodenman. In August 1975, Woodenman was widely distributed not only to military and federal agencies, but also to the computer industry and computer science research community. More than 100 review teams evaluated Woodenman (6:35).

the capability for many programmers to work together on the

Tinman. The response to Woodenman led to a complete set of requirements in January 1976 called Tinman. At this time, Tinman was officially approved for research and development efforts by the Assistant Secretary of Defense for Research and Development. Along with the development of Tinman requirements, sixteen companies performed evaluations of 23 programming languages against the developing requirements. The languages included, FORTRAN, COBOL, PL/1, HAL/S, TACPOL, CMS-2, CS-4, SPL/1, JOVIAL J3, JOVIAL J73, ALGOL 60, ALGOL 68, CORAL 66, Pascal, SIMULA 67, LIS, LTR, TRL/2, EUCLID, PDL2, PEARL, MORAL, and EL/1. The results of the evaluations concluded:

> 1. No existing language was suitable for use as a common high order language for DoD embedded systems.
>
> 2. A single language was desirable.
>
> 3. A new language should be developed from an appropriate base (2:16).

Although each was considered inappropriate as the required language, the evaluators recommended Pascal, ALGOL 68 and PL/1 as appropriate base languages (2:15-16).

Ironman. In January 1977, the Tinman requirements were updated into the Ironman document. While both documents satisfied basically the same requirements, Ironman was written in an organized language description and manual format, whereas Tinman was organized around general areas. Ironman was basically the specification around which contractors developed their proposed language designs (6:48).

Two independent studies conducted for the Management Steering Committee for Embedded Computer Resources between January and November 1977 concluded that hundreds of millions of dollars could be saved in DoD each year if a common language was developed (2:16).

DoD-1. Based on the evaluation of Tinman requirements, the HOLWG was directed to develop a common high-order language named DoD-1. The DARPA was assigned to award the design contract. Wanting a language with high quality, and a language to be accepted outside the defense community, DoD opted for an international design competition from which to select the design. The request for proposal (RFP) was submitted in April 1977, requesting designs for the high-order language. DARPA selected four contractors to continue the design. All four designs were Pascal based. The contractors involved were: SofTech, SRI International, Intermetrics, and Honeywell/Honeywell Bull. In the period February through March 1978, the designs were evaluated by 125 design review teams, and two designs were selected to proceed (Intermetrics and Honeywell/Honeywell Bull). During this next phase of the development, emphasis was placed on programming environments. A language in itself was not capable of improving software development without a suitable support system. In 1978, the HOLWG distributed the Sandman document which addressed the technical and managerial aspects of the programming environments. Based upon the

response to the Sandman document, the Sandman document was revised and released as the Pebbleman document. With emphasis on the programming environment, the HOLWG released the final language requirements in June 1978 called Steelman, which corrected all past deficiencies (2:17-18; 6:48).

A review of the final two designs was conducted in March through April 1979. In May 1979 Honeywell/Honeywell Bull was awarded the contract for the new design. The Honeywell team was out of France and was headed by Dr. Jean Ichbiah (2:18).

Ada. It was at this time that DoD-1 was named Ada. Ada was selected to honor the mathematician Lady Augusta Ada Byron (1815-1852), Countess of Lovelace. The Countess worked with Charles Babbage on his difference and analytic engines. She recommended how the engines could be programmed, thus is known as the first programmer (6:48).

Stoneman. A continuing area of concern was the programming support environment. The Stoneman document, which was a revision of the Pebbleman document, was the basis for a project which started in mid-1980 to resolve this area of concern. Support environments may be catagorized as closed-ended or open-ended. In a closed-ended environment, "the user is given a fixed set of tools that are presumably sufficient to meet all basic requirements. A closed environment cannot be altered or extended, short of re-issuing the environment by suppliers" (6:50). An open-

ended environment tool set can be modified or extended at any time to meet the needs of the user. Stoneman applied the open-ended environment approach (6:50).

APSE. The Ada Programming Support Environment (APSE) is based on the Stoneman model. Potential cost savings and quality software are inherent in an APSE. The following is a description of an APSE:

> The purpose of an APSE is to support the development and maintenance of application software throughout its life cycle, with particular emphasis on software for embedded computer applications. An important concept in an APSE is the data base, which acts as the central repository for information associated with each project throughout the life cycle (10:78).

The end result of a suitable APSE is the potential for portable and reusable tools and application software packages (13:8).

KAPSE. To ensure maximum compatibility and portability between APSE's, the Stoneman model requires all machine dependencies of the support environments to be contained in the Kernal Ada Programming Support Environment (KAPSE). The purpose of the KAPSE "is to interface the tools to the hardware" (13:8). According to Bruce Sherman, vice-president of planning for TeleSoft Inc., the KAPSE interface provides "common definition which the APSE, compiler, standard I/O packages and applications may use to request system services" (18:141). The KAPSE will allow the transportability of APSE's from one host system to another.

AJPO. In December 1980 the HOLWG transitioned into the joint service Ada Joint Program Office (AJPO). The responsibility of the AJPO was, "to manage the DoD Ada program by coordinating the military services' efforts to introduce Ada and Ada Programming Support Environment (APSE)" (13:5). The functions of the AJPO were to:

1. Maintain the Ada language standard.

2. Develop common-use training and education materials.

3. Validate Ada compilers.

4. Foster the use of Ada within the software community.

5. Develop Ada software tools to meet the common needs of the services and other DoD agencies (13:5).

ANSI Approval. Publication of the Reference Manual for the Ada Programming Language (Language Reference Manual (LRM)) was completed in July 1980. The LRM was republished in December 1980 by DoD as a military standard (MIL-STD 1815). Approval of Ada as an American National Standards Institute (ANSI) standard language occurred after canvassing potential implementors and users of the language. Based upon favorable results of the canvass, and after minor changes, Ada was approved as an ANSI standard on 17 February 1983 (13:7).

gram is to ensure non-divergent implementations of Ada. DoD has trademarked the Ada name, thereby limiting the use of the name Ada only to those compilers having been validated by the Ada Validation Office. To become validated, compilers must pass a test suite containing more than 1700 rigorous program tests (6:52; 13:9).

Though a considerable effort has been made in the development of Ada and its support environment to date, much work and research is still necessary in developing suitable APSE's and KAPSE's to fully realize the potential of the language.

Language Features

In the literature there are many reviews of the Ada language which praise its modern programming features. According to Peter Fonash, Deputy Director of the AJPO, "Ada is more than just another new language; by design it incorporates many features needed to support modern software engineering practices. An intrinsic principle of modern software engineering is the use of an automated environment that provides complete life-cycle software support" (13:7). Jean Sammet, manager of software development for IBM's Federal Systems Division, agrees with Fonash and remarks that "many of these features have appeared in the past but they haven't been put together in the same [effective] way" (10:62). The features of Ada are many and varied. The

language has borrowed features from other languages, but by no means is Ada comparable to another language. It looks like Pascal at first glance; Ada, however, is an enormously larger and more powerful language. The following sections outline some of the features of Ada and describe how they are important to the language.

Package Concept. Jean Ichbiah, a major influence and head of the Ada design team, believes "the package concept is the core and major contribution of Ada" (10:62). A package is a program unit in Ada which defines a collection of related entities (2:474). These entities may be constants, variables, types, subprograms or any legal Ada construct. When defined in a package, these entities may then be used with any other program unit with a simple 'with' clause. This facility provides Ada with a level of abstraction never before available in a high-order language.

Strong Data Typing. Typing is borrowed from Pascal and allows the specification of data types. Strong data typing can be illustrated by a single example. By defining a data type 'coin' with the values <1,5,10, and 25> representing cents, and another data type 'currency' with values <1,2,5,10, and 20> representing bill denominations, the compiler would generate an error message if the two types were mixed in an arithmetic operation. This feature reduces software costs by detecting operations on incompatible data types at compile time (17:76).

Block Structure and Separate Compilation. This feature, borrowed from ALGOL and FORTRAN, allows separate compilation of program units, particularly subprograms, packages, and tasks. The major benefit is simpler error detection since each module may be compliled and tested as each is built (10:72).

Tasking. This feature of Ada allows separate portions of a program to execute concurrently. With this flexibility, Ada can perform such real-time applications as robotics, communications, interactive graphics, and computer-aided design (10:72).

Exception Handling. This feature is borrowed from PL/1. This gives the programmer the facility to define, find and trap errors using standard Ada constructs. Exception handling allows the programmer to maintain control of program execution when a condition has occurred which would normally terminate execution of the program (as in division by zero). This makes programs more flexible and portable (10:72).

Benefits. One criticism of Ada is the complexity of the language. It may be a difficult language to master; the benefits, however, far outweigh the difficulties of learning the language. Long-term benefits of using Ada include:

1. Ada programs will be transportable, that is, a program written to run on one machine may be moved to another machine, recompiled, and executed with very few or no changes to the source code.

2. Packaging and separate compilation will allow programs to be constructed using existing modules. This, along with the use of exception handling, will make for reliable programs.

3. All of the above benefits will make programs written in Ada much easier to maintain (10:72).

Ada as a General Purpose Language

This section concerns the suitability of Ada as a general purpose programming language. It will address the views on the potential of Ada in the general purpose environment based on the features of the language as compared to other languages used in the same environment.

Advantages. While Ada was designed for embedded systems, members of the AJPO predict that eventually Ada will become a programming standard not only in the DoD, but also in the non-DoD community. Though the Ada design did not address the COBOL environment of financial and inventory management, nor the scientific environment of FORTRAN, the AJPO believes Ada is suitable for these environments. Fonash explains this attitude: "because Ada is a modern programming language that embodies good software engineering principles and modern language features, there appears to be a growing recognition that Ada is suitable for areas other than the embedded computer applications on which it was designed" (13:9). For these reasons, Ada should be a suit-

able language for the traditional COBOL and FORTRAN applications (13:9).

Commercial firms have demonstrated Ada's capability for use in business and non-DoD applications. Ralph E. Crafts, vice-president of Operations and Marketing at INTELLIMAC, Inc., has documented business and other non-DoD Ada applications. Examples of actual Ada business applications include:

1. A Multi-state Payroll System installed by a manufacturing facility in March 1982.

2. An Inventory and Parts Control System installed in June 1982.

3. An integrated General Ledger Accounting System installed in the summer of 1984 (5:70).

As a result of three years of Ada development for commercial applications at INTELLIMAC, Crafts feels there are many benefits in using Ada. He states, "The primary benefits to be realized from using Ada in the commercial environment are: enhanced utilization of structured analysis and design; the use of Ada as a PDL; accurate, functional deliverables; reuse of existing code; high productivity; and lower life cycle costs" (5:71).

According to Crafts, Ada provides a structured, engineering approach to software development. Language features such as, modularity and packages make it difficult to write poorly designed and unstructured programs (5:71).

The use of Ada as a PDL (program design language) is a benefit of the language. A PDL is usually a nonexecutable extension of a language which aids in the design of programs. This extension of the language is usually easy to read English statements, which enhances the readability of the program design. Ada alone can be used as a PDL in lieu of developing PDLs for aiding Ada program design. According to Crafts, there are many Ada projects which are using Ada as a PDL (5:71).

The use of structured design and programs written in understandable code benefit the delivery of accurate and functional programs to the end user. Crafts contends that the use of structured designs and code written in understandable English will result in the end user receiving the product specified and expected (5:71).

The reuse of existing Ada source code enhances productivity and reduces software costs. Two examples as explained by Crafts follows:

1. An order entry system program which consisted of 50,000 lines of Ada source code was developed several months ahead of schedule. Approximately 80 percent of the program reused existing Ada programs. Not only was time of program development reduced, but the time to test and debug the program was also reduced because only 20 percent of the software was new.

2. One programmer developed an 8000 line Ada program in one week by using existing Ada programs (5:71-72).

Even without the use of existing Ada programs, Ada enhances productivity. Over the three years of Craft's study Ada programmers have averaged 50 lines of operational code per day which includes design, testing, and debugging. This is a 800 to 900 percent increase in productivity over other languages. The Moog Company of Buffalo, New York had similar results. Prior to Ada, programmers typically wrote 200 lines of code per month. With Ada, they averaged 1200 lines per month (5:72, 6:54).

Life cycle software costs are reduced with Ada. Craft states in generalities, without citing figures, that Ada efficiencies of modifications, upgrades and changes to existing Ada programs makes maintenance of Ada software inexpensive (5:72).

One of Ada's advantages is the ability to handle large, complex software projects. Richard LeBlanc and John Goode of the Georgia Institute of Technology find Ada well designed for large complex systems. The structured programming design of Ada is based on the concept of modularity. Modularity allows the programmer to reduce large systems into smaller and easier to handle units. In Ada these units are packages, subprograms, and tasks. Modularity maximizes program reliability, readability, and maintainability (14:75).

Although the members of the AJPO and others are confident that Ada can sucessfully be applied to other than embedded applications, there are critics of the language.

Disadvantages. The most common criticism of Ada is the complexity of the language. LeBlanc and Goode consider this unfair treatment of Ada. Whereas Pascal is a relatively simple language, it is not designed for large-scale software development. The design goals of Ada and Pascal differ. Pascal was designed as an educational tool, whereas Ada was developed with a large range of objectives, to include large-scale projects. LeBlanc and Goode go on to say that the differences in design objectives should first be considered before being too critical of Ada (14:75,81; 21:248).

One may wonder whether the added features of Ada are necessary in a general purpose language. David Coar, a technical product staff member of Floating Point Systems Inc., conducted a comparison of Pascal, Ada and Modula-2. Pascal was designed by Nicklaus Wirth as an educational programming tool, suitable for modest size projects. Pascal is not recommended for major commercial or industrial projects, and for these reasons, Pascal has never been thought of as a true systems-implementation language. Modula-2 is Wirth's effort to go one step further than Pascal and design such a systems-implementation language. Modula-2 is a language with similar design goals as Ada. Examples of similar goals are; facilities for hardware interfacing, and

the capability for many programmers to work together on the same project. Coar's conclusions were that Modula-2 outperformed Pascal, and was better than most available languages. The extra features of Ada, in his opinion, were of marginal value as an implementation language (4:232).

III. METHODOLOGY

Introduction

Ada was primarily developed and designed for use in embedded systems. The primary objective of this research is to determine if Ada is suitable for use in general purpose programming applications. To determine Ada's suitability, this study compares Ada to two proven high order languages, FORTRAN and Pascal, in specific applications. The following sections describe the procedures used in this study.

Research Procedures

The choice of a methodology in this research was influenced strongly by the DPD. This document directed a study of Ada and specifically tasked AFIT to "use Ada in one or more selected applications and provide evaluation reports in pursuit of the stated objectives" (19:13). Following this guidance, this research compares non-embedded programs written in FORTRAN and Pascal, against the same programs translated into Ada.

Selection of Programs. The first step of this study consisted of selecting three programs from non-embedded applications written in the high order programming languages, FORTRAN and Pascal. The size of the programs were

relatively small, ranging from 60 to 300 lines of source code. This size was desirable for two reasons:

1. Time limitations. Since the study involves the translation of the programs into Ada, the researchers did not want to spend an excessive amount of time translating programs. Programs in the selected range of source code lines were determined to be appropriate and within the time available to complete the study.

2. Manageability. The researchers did not want to be overwhelmed by programs exceeding 1000 lines of code. The study is concerned with determining the suitability of Ada for use in the general purpose environment, not the researchers' ability to comprehend and translate large programs.

The sampling design used in the selection of the three programs would be classified as nonprobability sampling, that is each population element (i.e. possible programs to select from) does not have an equal chance of being selected. The objectives of the study justifies the use of this type of sampling technique. According to Emory "a random sample that is a true crosssection of the population may not be the objective of the research. If there is no desire to generalize to a population parameter then there is less concern about whether or not the sample is fully representative" (9:177). The objective of this study is not to generalize about Ada's superiority or inferiority to FORTRAN and

Pascal, but rather from the comparison with these languages, to determine the suitability of Ada to solve problems typically solved using these languages (9:176-177).

In selecting the programs to be evaluated, the particular nonprobability sampling method used was the method Emory defines as purposive (9:177-178). A judgment was made as to which programs were selected. The decision to use programs originally written in FORTRAN and Pascal was based on the researchers' experience and familiarity with these languages. The three programs selected for translation were:

1. A simulation program which simulates a single server, single queue system. Program title is TRUCK, and it is originally written in FORTRAN. (1:76-83)

2. A numerical computation program which approximates the area under a curve using the Trapezoidal Method of Numerical Integration. Program title is TRAP3, and it is originally written in FORTRAN. (15:207-208).

3. A menu driven, interactive text processing program which updates a library file system. Program title is LIBLIST, and it is written in Pascal (22:274-280).

These programs were judged by the researchers as having a wide variety of features which would test Ada's suitability as a general purpose programming language.

Translation

In translating a software package from one high-order language to another, two characteristics of the original are of prime importance for the translation to be considered correct. The first is execution equivalence, including functional equivalence and efficiency. The second is source code quality (8:3).

Exact execution equivalence would be for two programs, each written in a different high-order language and each compiled and linked, to contain the same number of machine instructions in the executable image file and to use the same amount of system resources at execution time. Execution equivalence for practical purposes is almost impossible to do. In this study, execution equivalence is defined as functional equivalence, that is, both the original and the Ada translation will produce the identical output given identical input. Also included here is efficiency. To the highest degree possible, given the limitations of the unvalidated Ada compilers, the Ada translations are as efficient as possible in terms of processor time and storage used (8:4).

The quality of the translated code is the other important characteristic to be considered during a translation. The code should be "readable, easily understandable, and embrace the style and intent of the language in which it is coded. Translations should also result in robust imp-

lementations, using to the fullest extent possible the power of the target HOL" (8:5).

To satisfy all of the above requirements, each original program under went two translations: a line-by-line translation and a complete redesign.

Line-by-line Translation. In this translation, the original was translated with a one-to-one correspondence between the original and the Ada code to the highest degree possible within the constraints of the language. Sections of code not translatable in this manner were functionally translated and annotated as such, maintaining the existing structure and flow as much as possible.

This type of translation is done to establish a baseline of functionality (i.e. identical output given identical input), and efficiency against which to compare the original programs and the complete redesign translations.

Complete redesign. In this translation the prime consideration with respect to the original is functionality. In order to use and exercise the large and powerful set of constructs available in Ada, the original problem is solved using object-oriented design, a design methodology described by Booch (2:40-44).

Measurements

Using the above mentioned translations, this study measures the differences and similarities between the trans-

lated and original programs. The following qualitative and quantitative measurements are made on the programs:

1. Functional Equivalence. The most important question to be answered in this study is to determine if the output from non-embedded applications can be replicated using Ada. The first measurement of this study compares the output generated by the original programs with the output of the Ada translated programs. To insure functional equivalence of the programs, it is essential that identical input be used for each related program.

To determine functional equivalence, this measure requires a qualitative assessment of the output. If the output is not identical, this study explains the consequences leading to the deviations. Specific areas addressed are the differences/similarities in I/O, real number precision and other factors which cause differences in the output between the original and translated programs.

2. Storage Efficiency. This measurement determines quantitatively the number of lines of source code necessary to replicate the original program's output, and the amount of storage space required in the runtime system. This measurement determines the storage space efficiency of Ada as compared to the other languages in the given applications. This measurement also involves a qualitative assessment of the reasons creating the differences, if any.

3. Execution Efficiency. This measurement involves comparing the execution runtimes of the original programs with those of the translated programs. This measurement determines the execution time efficiency of Ada as compared to the other languages in the given application.

4. Maintainability. This measurement is a qualitative assessment of the types of errors encountered while:

a. Debugging the original programs when initially running the programs on the Unix operating system.

b. Debugging the Ada translated programs. This measurement directly relates to the next measurement, which is transportability. In this study, transportability involves compiling the programs on different compilers.

5. Transportability. This measurement is a quantitative measurement of the number of changes required to compile both the original and translated programs on different compilers. Specifically the compilers used are:

a. Ada: TeleSoft, version 1.5, TeleSoft, version 2.2, and Verdix, version V04.06.

b. FORTRAN: Microsoft's FORTRAN-80, FORTRAN Extended Version 4 and the FORTRAN 77 compiler developed by Bell Laboratories, August 1978.

c. Pascal: Berkeley Pascal Compiler, Version 2.0.

6. Readability. This measurement is a qualitative assessment of the ease of understandability of the source code of the original and translated programs. This is demonstrated by selecting identical portions of the original and translated programs and allowing the reader to make his or her own assessment as to the readability of the compared source codes. Identical portions of a program is defined here as parts of a program performing an identical function, for instance reading a file, departing a queue, ect.

7. Miscellaneous. This study also records any findings important to the overall evaluation of the language in the chosen applications.

IV. FINDINGS AND ANALYSIS

Introduction

This chapter presents and analyzes the findings of this study. The Findings and Analysis Chapter presents the findings of the numerical computation application, the simulation application and the interactive text file processing application as they apply to the research questions proposed in chapter one. The original programs, translated programs, and the output for these programs which generated the data for these findings, are found in Appendices A through MM.

The program chosen to evaluate Ada's suitability in numerical computation applications was originally written in FORTRAN and is titled TRAP3. TRAP3 computes the approximate area under a curve described by a function defined in the program using the Trapezoidal Method. The original program consisted of a main program and one subroutine. The original TRAP3 program is found in Appendix A. The Ada line-by-line TRAP3 programs are found in Appendices B, C and D. The Ada redesign TRAP3 programs are found in Appendices E, F, G and H (15: 207-208).

The program chosen to evaluate Ada's suitability in a simulation application was originally written in FORTRAN and

is titled TRUCK. TRUCK is a simulation program that models a single server queue with interarrival times of Ø.33 per hour and service times of Ø.25 per hour. The program outputs a variety of results corresponding to the simulation. The original program consists of a main program and six subroutines. The original TRUCK programs are found in Appendices I and J. The Ada line-by-line TRUCK programs are found in Appendices K, L and M. The Ada redesign programs are found in Appendices N, O, Q and R (1: 76-83).

The program chosen to evaluate Ada's suitability in file processing was originally written in Pascal and is titled LIBLIST. LIBLIST is an interactive text file processing program which updates a library file system. The original program LIBLIST consists of a main program and five procedures. The original LIBLIST program is found in Appendix S. The Ada line-by-line program is found in Appendix T. The Ada redesign programs are found in Appendices U, V and W (22: 217-225).

The findings of this research are organized as they pertain to the research questions proposed on page five. In presenting the results of this research, the findings of the line-by-line translation are presented first, followed by the Ada redesign translation findings.

Functional Equivalence

Research question one addresses functional equivalence. In this research, functional equivalence defines the extent to which programs translated in Ada replicate the output of the original programs. The results of comparing the outputs generated during this research indicated that the Ada translated programs did replicate the original programs. However, there were slight differences and difficulties encountered during the research, and they are addressed below. Output listings for the original programs are found in Appendices X, DD, EE and KK. The Ada output presented below is that generated while using the Verdix compiler.

Output from the Ada line-by-line translation of the TRAP3 program resulted in virtually identical output when compared to the original FORTRAN program. The TRAP3 FORTRAN and Ada line-by-line translation outputs are in figures I and II.

```
Trapezoidal integration with end correction

      1  4.44444
      2  1.77535
      4  2.13427
      8  2.19111
     16  2.19675
     32  2.19719
     64  2.19722
    128  2.19723

Area =   2.19723
```

Fig 1. FORTRAN: TRAP3 Output

```
Trapezoidal integration with end correction

  1  4.44444444E+00
  2  1.70534979E+00
  4  2.13427396E+00
  8  2.19110817E+00
 16  2.19675417E+00
 32  2.19719294E+00
 64  2.19722256E+00
128  2.19722445E+00

Area = 2.19722445E+00
```

Fig 2. Ada: TRAP3 Output (Line-By-Line)

The only difference in the two TRAP3 outputs is the precision of real numbers used in performing numeric computations. From the output, Ada expressed nine digits of precision, whereas the FORTRAN output expressed six digits of precision.

This difference is easily rectified using the facilities of Ada. While FORTRAN does not allow designating the precision of real numbers, Ada has such a facility. By declaring in Ada, 'type six is digits 6;', the precision of objects declared as type six are constrained to six digits of precision. Therefore, executing the Ada program with objects of type six instead of type float, results in output identical to that of the FORTRAN program. The Ada output with six digit precision is in figure III.

```
Trapezoidal integration with end correction

          1  4.4444E+00
          2  1.70535E+00
          4  2.13427E+00
          8  2.19111E+00
         16  2.19675E+00
         32  2.19719E+00
         64  2.19722E+00
        128  2.19723E+00

Area = 2.19723E+00
```

Fig 3. Ada: TRAP3 Output (With 6 Digits Precision)

As a point of comparison, when using the 1.5 TeleSoft compiler, real number precision was eight digits, indicating that, default real number precision is implementation dependent. The source code listings for the Ada line-by-line translation using the TeleSoft-Ada compiler version 1.5 is in Appendix B. The output listing is in Appendix Y.

The Ada redesign, as with the line-by-line translation, resulted in output identical to that of the original. Output listings for the default FLOAT version and the Ada redesign six decimal digit version are given in Appendices BB and CC respectively. Output listings for the Ada line-by-line programs using the Vads compiler are given in Appendices Z and AA.

The FORTRAN TRUCK program and the Ada line-by-line translation resulted in similar output. Again, as in TRAP3 precision factors created a slight difference in the output. The TRUCK FORTRAN and Ada line-by-line outputs are in figures IV and V.

```
   TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER
QUEUE.


 DSEED =          0.56700000d+03
 MEAN ARRIVAL TIME(MIAT) =          0.3333
 MEAN SERVICE TIME(MSVT) =          0.2500

 PROPORTION OF TIME DOCK CREW IS BUSY =     0.73

 MAXIMUM LENGTH OF WAITING LINE =       15

 AVERAGE TIME TO TRANSIT SYS.     0.80    HOURS..

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM  0.01

 SIMULATION RUN LENGTH  509.36    HOURS..

 NUMBER OF TRUCKS UNLOADED =     1500

 NUMBER OF RANDOM NUMBERS USED =       3007

 AVERAGE NUMBER OF UNITS IN SYS.=       2.370


 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=    1207.228
(TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=         2.9547
```

Fig 4. FORTRAN: TRUCK Output

```
   TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      5.67000000E+02
 MEAN ARRIVAL TIME(MIAT) =       3.33333333E-01
 MEAN SERVICE TIME(MSVT) =       2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.30942508E-01

 MAXIMUM LENGTH OF WAITING LINE =          15

 AVERAGE TIME TO TRANSIT SYS. 8.04808941E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 5.66666667E-03

 SIMULATION RUN LENGTH 5.09356313E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =        1500

 NUMBER OF RANDOM NUMBERS USED =        3007

 AVERAGE NUMBER OF UNITS IN SYS.=     2.37007647E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
1.20721341E+03     (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.95470962E+00
```

Fig 5. Ada: TRUCK Output (Line-By-Line)

The precision problem created a few difficulties in replicating the FORTRAN TRUCK output into the Ada output. In the TRUCK program it was necessary to generate random numbers. To have identical output, the random number string generated for the original and translated programs had to be identical. To insure this, the original TRUCK program was modified to include a random number generator subroutine 'gguos' which was common to both programs. However, even

with an identical 'ggubs' subroutine, the precision difference in performing operations of real numbers created two different strings of random numbers, thus resulting in two different outputs. This was rectified as in TRAP3 by designating the digits of precision in the Ada subroutine 'ggubs' to six, therefore the random number strings were identical, resulting in similar output. The 'ggubs' subroutines are found in Appendices I and L.

Another difference in the outputs is that the number of digits of precision actually in the FORTRAN output can be limited by the formatting features of FORTRAN. The FORTRAN formatting statement allows the designation of the number of digits to be output for real numbers and integers. Ada lacks the necessary library packages needed to duplicate this feature of FORTRAN's formatting capability at present. However the same results could be generated using Ada by declaring types with the required precision and by then explicitly converting the values to the designated type prior to outputting. For example,

```
type six_digits is digits 6;
sum : float;
begin
sum := 1.0/3.0;
new_six := six_digit(sum);
```

in this case sum is expressed as 0.33333333E00 while new_six is expressed as 0.33333E00.

The output of the Ada redesign of TRUCK experienced the same problem of generating identical random number streams as did the line-by-line translation. The problem was overcome in the manner described above.

While the line-by-line translation generated output almost identical to that of the original FORTRAN version, two of the performance measures generated by the Ada redesign consistantly deviated from the line-by-line translation and original by one.

One of the measures which differs is the maximum queue length. The deviation is due to differences in the maintenence of each programs' respective queue.

When an arrival is generated in the Ada redesign program the queue length index is bumped and the arrival time is stored in the queue at the position given by the index irrespective of the condition of the queue. The index, therefore, always points to the last element in the queue. The FORTRAN original and line-by-line translation check the condition of the queue and if the queue is empty then the arrival time is hard-coded into the first element in the queue but the the queue length index is not incremented. The index is bumped only when the queue is not empty; the index is always one less than the number of elements in the queue. To prevent the last element in the

queue from being overwritten, the program indexes the queue with a local variable assigned the value of the sum of the incremented queue length index and tne queue busy flag. The relevant code is shown in figure VI.

```
          IF(LST.EQ.1) GO TO 20
          LST=1
          CHKOUT(1)=CLOCK
          ...
          GO TO 100
20        LQT=LQT+1
          I=LQT+LST
          ...
          CHKOUT(I)=CLOCK
          ...
          IF(LQT.GT.MQ) MQ=LQT
          ...
100       ...
```

Where

LST	=	Queue busy flag.
CHKOUT	=	Queue.
CLOCK	=	Current system time (arrival time).
LQT	=	Queue length index.
MQ	=	Maximum queue length.
I	=	Local variable.

Fig 6. Check of Queue Condition

Since the queue length index is always one less than the actual number of elements in the queue, the maximum queue length will be likewise. Therefore, the original FORTRAN code which indexes the queues is in error, and the Ada redesign program properly indexes the queue. This explains the difference in the maximum queue length output.

The other difference in output between the Ada redesign and the original is in the measure of the number of random numbers used in the simulation. The Ada redesign is apparently always one low. In fact, the number given by the redesign is the actual number used in both programs. The apparent deviation is due to differences in the way each random number is picked from the stream in the two programs.

The redesign initializes the count to zero and bumps it immediately before the number is used, while the original initializes the count to 1 (the array index) then bumps it immediately after the number is used. The index will always be pointing to the next (unused) random number in the stream and will be one high at the conclusion of each iteration of the simulation.

Output listings from the Ada redesign translations of TRUCK are in Appendices II and JJ. The Ada line-by-line output listings are in Appendices FF, GG and HH.

A comparison of the output for the Pascal LIBLIST and Ada line-by-line LIBLIST is identical. No difference is evident in the manipulation of output between the two programs. A sample of the Pascal LIBLIST and Ada line-by-line LIBLIST output is in figure VII.

```
WAR AND PEACE
LEO TOLSTOY
        100
TOM SAWYER
MARK TWAIN
        200
INTRODUCTION TO PASC
RODNAY ZAKS
        300

END OF LIBRARY FILE
```

Fig 7. Sample of LIBLIST Output

The only differences between the output of the Ada redesign of LIBLIST and the Pascal and the line-by-line translation are cosmetic: the format of the menu and output listing are different. Though the line-by-line translation and the redesign use different techniques to accomplish the objective, (linked-list using access types versus chained list) all three programs will insert and delete a book from the list, as well as print the list in ascending order by call number. Source code listings for each of the LIBLIST programs are in Appendices S, T, U, V and W. Output listings for each of the LIBLIST programs are in Appendices MM, LL and MM.

Storage Efficiency

Research question two addresses the degree of difference in the lines of source code and the size of the executable code files required to duplicate the output

between the translated and original programs. As a note of clarification, for this study a line of source code is defined as follows:

1. A carriage return depicts a line of source code.

2. Comment lines and blank lines are not counted as a line of source code.

In comparing the Ada line-by-line programs against the original programs, the Ada programs required more lines of code to replicate the FORTRAN output, and virtually the same number of source code lines to replicate the Pascal output. The number of lines of code required in the Ada and the original programs are outlined in Table I.

TABLE I

Lines of Source Code

	TRAP3	TRUCK	LIBLIST
FORTRAN	37	218	--
Pascal	--	--	197
Ada (line)	59	253	201
Ada (redesign)	98	313	209

The primary areas which created the differences in the required source code in the FORTRAN versus Ada line-by-line code were:

1. Declaration and Initilization of Objects. In the Ada TRUCK program, the main program required the declaration of 22 objects and three types within the specification part of the program. This was necessary so that the objects could be passed as arguments and be visible in the subroutines. The FORTRAN program on the other hand used two common statements to make the objects visible within the subroutines. Quantitatively the specification part of the Ada line-by-line TRUCK program had 21 lines of code, while the declarative portion of the FORTRAN program had 12 lines of code.

Although FORTRAN can declare and initialize variables in fewer lines of code, and in fact does not require the explicit declaration of variables, Ada requires the declaration of all types and variables which aids in the maintainability, understandability and the capability to debug programs.

2. Language Feature Differences. A few differences in the type of features the FORTRAN and Ada languages support, created a few lines of difference in the coding. Since Ada does not have the FORTRAN feature of statement functions, the line-by-line translation required two additional functions in the TRAP3 program. The FORTRAN TRAP3 used the two statement functions:

```
F(X) = 1.Ø/X    and   DF(X) = -1.Ø/(X*X)
```

whereas Ada necessitated writing two functions to duplicate the above statements. The two additional functions required ten additional lines of code, i.e. inclusion of specification and body parts, and the return statement. This difference can be seen in Appendices A and C.

In the main body of the FORTRAN TRUCK program the conditional goto and the goto statement was used four times. Ada does have a goto feature, however, since Ada does not promote the use of goto statements, it was not used in the TRUCK line-by-line translation. Instead, an if-then-else structure was used to replicate what the original program was doing with the goto statements. Using the if-then-else created additional lines of code over the goto.

To demonstrate the goto in Ada, it was used in the TRAP3 program. Appendix C demonstrates the use of the goto. It basically operates in the same manner as in FORTRAN, however with limitations. The scope of an Ada goto is limited in that the execution of a goto can not transfer control into a compound statement such as an if, loop, accept, case, block, or accept statement, i.e. Ada gotos may transfer control only within the same lexical level.

3. I/O Differences. The single largest difference in the number of lines of code required to replicate the original FORTRAN outputs involved differences in output facilities. The FORTRAN provision of formatting

allowed the FORTRAN programs to output results in fewer lines of code than the Ada line-by-line programs. It required 22 additional lines of Ada TRAP3 code and 35 additional lines of Ada TRUCK code to replicate the original output in Ada. Reasons include; attaching the TEXT_IO package, instantiating the generic float_io and integer_io packages, and the number of put and new_line statements required. Appendices A, C, I and L show the I/O differences.

As is shown in Table I the Ada redesigns required significantly more lines of code than the FORTRAN originals. In addition to the reasons given for the differences between the originals and the line-by-line translations, two others account for the additional lines needed to duplicate the original Ada.

First, in both the TRUCK and TRAP3 programs, the Ada redesign encapsulated type and subprogram definitions in packages. This construct requires each subprogram specification to be entered in the package specification as well as the package body, resulting in code redundance.

Another cause of the increase is the way the code was assembled to enhance readability. This accounted for most of the additional code.

Included here would be individual object declarations, even for objects of like type. An example might be:

```
UPPER_BOUND   : FLOAT;
LOWER_BOUND   : FLOAT;
TOLERANCE     : FLOAT;
AREA          : FLOAT;
```

rather than

```
UPPER_BOUND, LOWER_BOUND, TOLERANCE, AREA : FLOAT;
```

Another technique was to break long lines of code into smaller, more readable lines such as:

```
AREA  :=  (F(LOWER_BOUND
        +  F(UPPER_BOUND))
        *    (UPPER_BOUND - LOWER_BOUND)
        /  2.0;
```

rather than

```
AREA := (F(LOWER_BOUND + F(UPPER_BOUND)) * (UPPER_BOUND
        - LOWER_BOUND) / 2.0;
```

There was very little difference between the Pascal, Ada line-by-line, and Ada redesign LIBLIST programs. This could be expected since Ada is a Pascal based language. The Pascal program required 201 lines of code, the Ada line-by-line required 197, and the redesign required 209. The difference being accounted for by instantiating the integer output package in Ada and the use of packages in the Ada redesign.

To compare the size of the executable code files the 'ls -al' command on the UNIX operating system was used. This command shows the size of the executable code files in bytes. In all cases, the Ada code required more bytes of storage space for the executable code. Table II shows the size of the files required.

TABLE II

Size Of Executable Code Files

	TRAP3	TRUCK	LIBLIST
FORTRAN	36864	40960	--
Pascal	--	--	26624
Ada (line)	63488	69632	64512
Ada (redesign)	64512	68608	68608

Probable reasons for the size discrepancy of the executable code is the refinement of the compilers. The early versions of the Ada compilers are obviously less efficient, as can be seen by comparing the Pascal LIBLIST and the Ada line-by-line LIBLIST programs. The two mentioned programs are of the same relative size and are performing the same functions, however the size difference of executable code files is quite considerable.

Execution Efficiency

Research question three addresses execution efficiency. The Unix 'time' command was used to find the CPU times. Running each of the programs five times each resulted in the average execution times as shown in Table III.

TABLE III

Execution Times

	TRAP3	TRUCK
FORTRAN	0.02 sec	0.67 sec
Ada (line-by-line)	0.04 sec	8.12 sec
Ada (redesign)	0.083 sec	5.18 sec

The Ada program in all cases required more CPU time to execute the programs. The probable reason for the difference is that the FORTRAN compiler is a more refined, more advanced generation compiler, whereas the Ada compilers are virtually in their infancy.

The CPU runtime of the Pascal LIBLIST programs were not recorded. Due to the nature of the Pascal LIBLIST program, i.e. an interactive text file processing program, finding execution times did not appear to be of any relevance to this research. However, in executing the programs there was not any noticeable differences in response time between the Ada and Pascal programs.

Maintainability and Transportability

Research question four compares the maintainability and transportability of Ada programs against that of the selected programs. The manner in which this question was handled

was by first describing the problems encountered and the actions that were necessary to compile and execute the original programs on the UNIX system with the available facilities (compilers). Second, describe any differences or problems encountered in translating the original programs, and third, address the problems faced in compiling and executing the Ada programs on the different Ada compilers available for this research. By the description of the above actions, subjective conclusions can be drawn on this important feature of a programming language.

The original TRAP3 program was taken from FORTRAN for Scientists and Engineers by Alan R. Miller. The original program was compiled with Microsoft's FORTRAN-80, Version 3.4 compiler.

The original TRAP3 program called for passing two statement functions from the main program as arguments to a subroutine. The FORTRAN 77 compiler used in this study would not allow such an operation. The changes necessary to compile and execute the program involved placing the statement functions directly within the subroutine. Therefore one major change was required to compile and execute the original TRAP3 program.

The Ada line-by-line translation of TRAP3 was originally compiled on the TeleSoft 1.5 compiler. In translating to Ada, the only difference encountered in the coding was the lack of statement functions in Ada. The Ada trans-

lation as mentioned before, required writing functions to represent the FORTRAN function statements.

The output of the TRAP3 program run with the Verdix compiler was slightly different because of the size of the largest integer. The output was shifted five spaces to the right on the screen because integer last under Verdix is 2147483647 whereas it is 37567 under TeleSoft 1.5., indicating implementation dependence.

The result of this is that the FORTRAN TRAP3 program required one syntax change in order for the program to compile and run, whereas the Ada TRAP3 program when moved from the TeleSoft 1.5 to the Verdix compiler had zero syntax errors and successfully compiled and executed.

The original TRUCK program was taken from Discrete-Event System Simulation by Banks and Carson, and modified and compiled with the FORTRAN extended version 4 compiler. The original TRUCK programs are found in Appendices I and J.

When the FORTRAN TRUCK program was transported to the UNIX system and compiled with the FORTRAN 77 compiler, the following conditions existed:

1. The original program made access of the IMSL library subroutine 'GGUBS' to generate random numbers. When transported to the UNIX ASC system initially, IMSL was yet to be implemented, therefore the program could not compile. The program was modified by writing a random number generator subroutine called GGUBS also.

2. The original FORTRAN program consisted of 3000 customers (trucks) and an array of dimension 6500 with real number elements (random numbers). Due to a storage problem with the Ada compiler, which will be explained later, the TRUCK problem was reduced to have 1500 customers and the need for only 3500 random numbers. To modify the FORTRAN TRUCK program to include the lesser number of customers and random numbers required one change involving the number of customers (NCUST=3000 to NCUST=1500), and eight modifications were necessary to change the number of random numbers required and the dimension of the random number array (NR=6500 to NR=3500 and seven changes to R(6500) to R(3500) which was in each of the common blocks).

In contrast, modifying these two changes in the Ada line-by-line TRUCK program required only two modifications to the program (NR:=3000 to NR:=1500 and by declaring an array type 'type RN is array (integer range 1 .. 6500) of float;' only requires the 6500 be changed to 3500 once.

The Ada line-by-line TRUCK program was initially compiled with the TeleSoft 2.2 version. The following problems were encountered with the Ada line-by-line program in developing successful output:

1. The TeleSoft 2.2 compiler severly restricted the array size. Although the Ada TRUCK program with an array dimension of 6500 would compile, it would not execute. The following execution error was raised: "Storage Error

(Sec_Stock_Overflow) Raised in Main Unit on Line # 83." Line #83 pointed to the subroutine 'GGUBS'. In trying different array sizes, the largest dimension the TeleSoft 2.2 would support was an array of size 622.

2. Compiling the same program with a 6500 dimensioned array with the 1.5 TeleSoft compiler resulted in the following compilation error: "Error: Data Size of Seg: 1 Proc: 1 is too big." Using different array sizes, the dimensional size of approximately 4000 was the extent the 1.5 TeleSoft compiler could support. Therefore the original TRUCK program was modified to use 1500 customers and a 3500 dimensioned array of random numbers.

3. Truncating versus rounding of numbers created problems in maintaining the Ada program. This problem was encountered in generating the random number string. The random number generator alogrithm required the explicit conversion of real numbers into integers. In the expressions:

```
X = 2.7762
Y = INTEGER(X)
```

the FORTRAN 77 compiler truncates the value to 2. The Ada 1.5 TeleSoft compiler rounded the value to 3. Therefore the Ada 'GGUBS' subroutine required modification. However, when using the Verdix compiler, the above Y value is truncated to the value 2, so again the 'GGUBS' subroutine required modifications again. All of this indicated that the explicit

conversion from real to integers is implementation dependent.

Converting from the TeleSoft 1.5 to the Verdix required modifying the random number generator as mentioned above. Other minor differences experienced included getting warnings for objects passed as arguments in subroutines without being initialized, and the need to attach a package with the natural log function for the service time and arrival time alogrithm. Natural logs are an intrinsic function of FORTRAN, however such functions are not standard in Ada. Therefore, a natural log function was written and encapsulated within a package. The natural log function is found in Appendix P.

The original LIBLIST program was taken from Introduction to Pascal, Including USCD Pascal by Rodnay Zaks. The LIBLIST program involved processing a library file to include inserting and deleting records to a text file. The program was modified to include a procedure for convenience which involved viewing the entire library file interactively. The original Pascal LIBLIST is found in Appendix S.

In transporting the Pascal LIBLIST program to the UNIX system with the avaliable Pascal compiler, only one major change was necessary to compile and execute the program LIBLIST. The original LIBLIST program as extracted from the text was typed entirely in upper case. The Pascal compiler

would not compile keywords, types and filenames entered in upper case.

Since the TeleSoft 1.5 compiler does not support generics and the TeleSoft 2.2 compiler was removed from the operating system the research was conducted on, the Ada LIBLIST program used the Verdix compiler exclusively.

In writing the Ada line-by-line LIBLIST program, there were no portions of the Pascal program which could not be duplicated due to the similarity of Ada and Pascal. The Ada and Pascal code was very similar. The original program used access types to link the library files in numerical order. Ada has incorporated the access type feature, and with very few syntax differences functions exactly the same as the Pascal access type.

In the Ada redesign effort, all three original programs were translated, compiled, and executed using the Verdix compiler. However, since the Telesoft-Ada compilers were removed from the system upon installation of the Verdix compiler, only the TRUCK program was compiled with the Telesoft software. No comments can be made regarding the transportability of the redesigned TRAP3 and LIBLIST programs.

When transporting the Ada redesign of TRUCK from the Telesoft to the Verdix compiler no code changes were required in the main (calling) program and only the instantiation of FLOAT_IO and INTEGER_IO was required to successful-

ly compile the SIMULATION_ROUTINES package. As with the movement of the line-by-line translation from Telesoft to the Verdix compiler, the same changes for the same reasons were necessary in the redesigned program for proper execution with the Verdix.

Source Code Readability

Research question five addresses source code readability. The readability of Ada code is hailed as one of the language's key features. This section presents the findings on the differences between Ada source code and the original programs' source code.

While Ada affords the programmer a rich set of tools with which to compose very readable code, it is apparent after even a cursory inspection of the Ada line-by-line translations that it is possible to write bad code in Ada. The Ada redesign effort was to translate the original code into Ada using all of the language features necessary to produce structured, readable and functionally equivalent code.

In addition to using the built-in features of Ada designed to enhance structure and readability, the programmer used the following conventions in coding the redesigned programs:

1. Individual object declarations.
2. Grouping of objects of like type at declaration.

3. Vertical alignment of the colon (:) and assign symbol (:=) at object declaration/initialization.

4. Indent all code between program unit and begin clauses and between begin and end clauses.

5. Follow standard rules of indention for loop, if, and case structures.

6. Vertical alignment of the goes-into symbol (=>) in subprogram specifications, calls, and case structures.

7. Vertical alignment of the assign symbol (:=) when possible in lists of assignment statements.

8. Use of object names as meaningful as possible.

9. Use of lower-case for all Ada reserved words and attribute invocations, and upper-case for all object names and type marks.

By following these rules as closely as possible the Ada redesign effort achieved significant improvements in readability and understandability over the original code. One of the best examples of this improvement is the difference between the calling programs of the FORTRAN and Ada redesign of TRUCK. The FORTRAN code is shown in figure VIII.

It is apparent that the author of this code had no concern for the readability of the software as none of the rules listed above were followed. Even the comments, rather than enhance the readability of the code, tend to clutter the code.

```
C     TRUCK PROBLEM-VARIANT OF PP 77-82 IN BANKS AND CARSON.
      PROGRAM TRUCK
      REAL MIAT,MSVT
      INTEGER NR
      COMMON /SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON /TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
      II=1
      DSEED=567.0
    1 NUMEVS=2
      MIAT= 1.0/3.0
      MSVT=.25
      NCUST=1500
C     WE WILL USE GGUBS TO GENERATE A STRING OF RANDOM #'S
C     ROUTINE GGUBS
      NR=3500
      CALL GGUBS(NR)
C     IIR WILL INDEX THE RANDOM NUMBER GENERATOR.
      IIR=1
C     CALL INITIALIZATION ROUTINE
      CALL INITLZ
C
C
C     CALL TIME-ADVANCE ROUTINE TO DETERMINE IMMINENT EVENT
C     AND ADVANCE CLOCK TO THE IMMINENT EVENT TIME.
   30 CALL TIMADV
C
C     VARIABLE "IMEVT" INDICATES THE IMMINENT EVENT.
C     IMEVT=1 FOR AN ARRIVAL.
C     IMEVT=2 FOR A DEPARTURE.
      GO TO(40,50),IMEVT
   40 CALL ARRVL
      GO TO 30
C     CALL DEPARTURE ROUTINE
   50 CALL DPART
C
C     CHECK TO SEE IF SIMULATION IS OVER. IF NOT RETURN TO
C     #30
    8 IF(ND.LT.NCUST) GO TO 30
      IF(II.EQ.1) DSEED=567.0
      ...
      IF(II.EQ.10) DSEED=2717.0
      CALL RPTGEN
C     WHEN SIMULATION OVER GENERATE REPORTS.
      STOP
      END
```

Fig 8. Sample of FORTRAN Source Code

```
with SIMULATION_ROUTINES ; use SIMULATION_ROUTINES;

            procedure TRUCK_SIMULATION is
   MEAN_INTER_ARRIVAL_TIME : FLOAT          := 1.0/3.0;
   MEAN_SERVICE_TIME       : FLOAT          := 0.25  ;
         STATS                        : STATISTICS;
           SERVICE_QUEUE                : QUEUE;
    RANDOM_NUMBER             : RANDOM_NUMBER_RECORD;
                        begin
              while STATS.REPETITION < 10 loop
                      INITIALIZE (STATS,
                                  SERVICE_QUEUE,
                             MEAN_INTER_ARRIVAL_TIME,
                                  RANDOM_NUMBER);

     while STATS.TOTAL_DEPARTURES < 1500 loop
        if STATS.NEXT_ARRIVAL < STATS.NEXT_DEPARTURE then
           GENERATE_ARRIVAL (STATS,
                             SERVICE_QUEUE,
                             MEAN_INTER_ARRIVAL_TIME,
                             MEAN_SERVICE_TIME,
                             RANDOM_NUMBER);
        else
           GENERATE_DEPARTURE (STATS,
                               SERVICE_QUEUE,
                               MEAN_SERVICE_TIME,
                               RANDOM_NUMBER);
        end if;
     end loop;
     GENERATE_REPORT (STATS,
                      SERVICE_QUEUE,
                      MEAN_INTER_ARRIVAL_TIME,
                      MEAN_SERVICE_TIME,
                      RANDOM_NUMBER);
  end loop;
end TRUCK_SIMULATION;
```

Fig 9. Sample of Ada Redesign Source Code

The Ada redesign version is shown in figure IX. The difference is striking. The code is clean, understandable, structured, and functionally equivalent. The problem of

overly long argument lists, which may have been a concern due to Ada's lack of the 'common' statement, was overcome by building record types of related data objects (RANDOM_NUMBER_RECORD, STATISTICS, QUEUE) and passing the record objects. While each of the rules listed above as well as features built into the language (end if, end loop, begin/end etc.) enhance the readability of Ada code, the most significant feature of Ada with regard to readability is the capability to create meaningful type and object names.

Again, the TRUCK program provides a good example. The FORTRAN and Ada code which initialize variables before each iteration of the simulation is shown in figures X and XI.

The lack of meaningful variable names in the FORTRAN routine due in part to FORTRAN being limited to variable names not exceeding six characters, makes the code very difficult to follow. To translate the code the Ada programmer was forced to use the strings printed in the report generating routine to decipher many of the names. The Ada code, however, with the use of meaningful names, is easy to follow and leaves the reader with little doubt as to the use of a given variable.

```
      SUBROUTINE INITLZ
      REAL MIAT,MSVT
       COMMON /SIM/ MIAT,MSVT,NCUST,LQT,
     1LST,TLE,CHKOUT(100),B,MQ,S,F,ND,
     2IIR,R(6500),DSEED
      COMMON /TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
C
C     SET SIMULATION CLOCK TO ZERO.
C     ASSUME SYSTEM IS EMPTY AND IDLE AT TIME ZERO.
C     INITIALIZE CUMULATIVE STATISTICS TO 0.
      CLOCK=0.0
      IMEVT=0
      LQT=0
      LST=0
      TLE=0
      B=0
      MQ=0
      S=0
      F=0
      ND=0
C     GENERATE TIME OF FIRST ARRIVAL,IAT, AND SCHEDULE FIRST
C     ARRIVAL
C     IN FEL(1)K.SET FEL(2) TO "INFINITY" TO INDICATE THAT A
C     DEPARTURE IS NOT POSSIBLE WHILE THE SYSTEM IS EMPTY.
      RR=R(IIR)
      X= -log(RR)
      X=X*MIAT
      XXT=1.0
      FEL(1)=CLOCK + X
      FEL(2)= 1.0E+30
      IIR=IIR+1
      RETURN
      END
```

Fig 9. Sample of FORTRAN Declaration and Initialization of Objects

```
procedure INITIALIZE
          STATS                      : in out STATISTICS;
          SERVICE_QUEUE              : in out QUEUE;
          MEAN_INTER_ARRIVAL_TIME    : in     FLOAT;
          RANDOM_NUMBER              : in out
                                       RANDOM_NUMBER_RECORD) is


   ARRIVAL_TIME   : FLOAT;
begin
   STATS.REPETITION             := STATS.REPETITION + 1;
   RANDOM_NUMBER.DSEED          := RANDOM_NUMBER.SEEDS
                                   (STATS.REPETITION);
   RANDOM_NUMBER.COUNT          := 0;
   RAN (RANDOM_NUMBER);
   ARRIVAL_TIME          :=  MEAN_INTER_ARRIVAL_TIME
                           * (-LN (RANDOM_NUMBER.NUMBER));
   STATS.CLOCK                  := 0.0;
   STATS.TIME_LAST_EVENT        := 0.0;
   STATS.SERVER_BUSY_TIME       := 0.0;
   STATS.TOTAL_TIME_IN_SYSTEM   := 0.0;
   STATS.TOTAL_ARRIVALS         := 0  ;
   STATS.TOTAL_DEPARTURES       := 0  ;
   STATS.MAX_Q_LENGTH           := 0  ;
   STATS.FOUR_HOURS_IN_SYSTEM   := 0  ;
   STATS.NEXT_ARRIVAL           :=  STATS.CLOCK
                                  + ARRIVAL_TIME;
   STATS.NEXT_DEPARTURE         := 1.0e30;
   SERVICE_QUEUE.LENGTH         := 0;
   SERVICE_QUEUE.IS_IDLE        := TRUE;
end INITIALIZE;
```

Fig 11. Sample of Ada Declaration and Initialization of Objects

Other Findings

Research question six is included as a catch-all to allow the discussion of any relevant finding not enumerated in the previous five questions. One such finding was uncovered during the Ada redesign of the LIBLIST program.

With Ada's capability to encapsulate data types in packages and with almost no restrictions on type names, it is not inconceivable that identical type names, even identical type definitions are in more than one package. This would not present a problem unless more than one of these packages were simultaneously imported by another program and an object of the type in question is declared in the using program. This situation arose during the Ada redesign of LIBLIST.

LIBLIST maintains a direct-access file of records stored on disc. Each record contains several fields each with information for a given book. One of the fields stores the call number of the book. Another field contains a pointer which links the records such that when read and printed while stepping through the chain, the records will be in ascending order by call number.

The pointer contains the position in the file of the next record in the chain. When these pointers are used to read or write records on the file, they must be converted to a type required by DIRECT_IO. That type is defined as:

```
type COUNT          is 0..implementation defined;
type POSITIVE_COUNT is 1..COUNT'last;
```

In addition to importing DIRECT_IO for file access, the program uses TEXT_IO to print prompt strings to the screen. Unknown to the programmer, TEXT_IO has an identical type definition to that shown above:

```
type COUNT          is 0..implementation defined;
type POSITIVE_COUNT is 1..COUNT'last;
```

The problem was not so much the ambiguity seen by the compiler, since both packages were directly visible and the package prefix notation was not used, as was the esoteric error message given by the compiler to flag the error.

The message, "identifier undefined," initially led the programmer to believe that the package containing the ambiguous type name was not visible. It was by accident that the programmer found the type defined in both DIRECT_IO (the one used in LIBLIST) and TEXT_IO. The ambiguity was resolved by using the package name prefix notation when referencing the type in the importing program. This illustrates the indiscriminate use of the 'use' clause.

V. CONCLUSIONS AND RECOMMENDATIONS

Conclusions

The primary objective of this research was to evaluate Ada's suitability in non-embedded applications. A comparison of the Ada translated programs against the original programs, indicated that Ada was a suitable programming language for the chosen applications.

The research demonstrated that the Ada translated programs did replicate the output of the original programs. Chapter 4 explains a few of the features in Ada, such as designating real number precision, which facilitates the replication of the output.

Although the findings show that more lines of source code were required as compared to FORTRAN, it was not in our opinion a substantial difference. As explained in Chapter 4, in the Ada redesign programs the increased use of source code contributed to the overall readability of the code. Concerning the input/output source code differences, the writing of an Ada I/O formatting package to be used with any Ada programs would eliminate that difference. Concerning Pascal, since Ada is a Pascal based language, the Pascal and Ada line-by-line programs were very similar.

The findings show that the FORTRAN programs run more efficiently than the Ada translated programs. Also the

FORTRAN and Pascal programs required considerably less space in executable code files. The probable reason for this is the degree of compiler refinement. Upon Ada compiler advancement, these deficiencies may be overcome.

It was difficult to draw any conclusions concerning the maintainability and transportability of source code. Although none of the FORTRAN or Pascal programs would compile or execute when moved to the UNIX operating system and using the available compilers, the Ada programs when used with different compilers also experienced problems. The Ada programs experienced no syntax errors when transported between compilers, but did experience difficulties due to implementation dependent features. For example, explicit conversion of real numbers to integers resulted in two different values depending upon which compiler was used. The TeleSoft 1.5 compiler rounded the real number, while the Verdix compiler truncated.

One important maintainability issue was raised in the findings of the TRUCK program, and that involved the amount of changes required to modify values of parameters in the programs. It was shown that changing two parameters like the number of customers and the dimension of an array in the FORTRAN TRUCK program required eight changes, while the Ada TRUCK program only needed two changes.

This difference is primarily due to the strong typing requirements of Ada. Strong typing can significantly in-

crease the maintainability of a program written in Ada when compared to an equivalent program written in FORTRAN.

The capability to write readable code in Ada was demonstrated by this research. However, the production of readable code does require a conscious effort on the part of the Ada programmer. The Ada line-by-line programs showed little or no improvement in readability, however, the Ada redesign programs using meaningful object names and types, sound program structure, and a few other simple programming techniques, demonstrates the degree of readablility improvement achievable with Ada.

This research did demonstrate that Ada could replicate the output of the three chosen non-embedded applications. The objective of replicating the output was achieved, however results from other areas examined such as execution times, and storage requirements proved disappointing. The authors feel that the results from these areas can be improved through the use of mature Ada compilers, and increased programmer experience with the Ada language.

This research covered a wide range of major areas which influence the performance of a programming language. Due to the range of areas examined, an in-depth examination of each of the areas was not possible. These areas need to be examined more in depth.

Recommendations

Upon completion of this research it was evident that more research is required on the Ada programming language. This research was limited to three non-embedded applications. Research in other non-embedded application areas is necessary to fully evaluate Ada's suitability in non-embedded applications. It is also necessary that validated and more mature compilers be used in any future studies. Only once this is accomplished can a decision be made concerning the ability of Ada to become the single DoD common programming language for all application areas.

This research covered a range of language features. A close examination of the maintainablity and transportability of Ada source code needs to be accomplished. These areas are essential for the evaluation of Ada as a common DoD language.

This research did not evaluate the COBOL programming language against that of Ada. To determine Ada's suitability in business applications, an evaluation of Ada against COBOL applications would be beneficial.

Finally, the attributes of Ada were not addressed. Language attributes appear to be one of the strong points of Ada. A study of the advantages of Ada's attributes versus features of other languages implementing similar capabilities will provide a more complete evaluation of the Ada language.

APPENDIX A

SOURCE LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
ORIGINAL FORTRAN

```
c       PROGRAM TRAP3
c
        REAL SUM, UPPER, LOWER, TOL
        DATA LOWER/1.Ø/, UPPER/9.Ø/, TOL/1.ØE-5/
c
c       f(X) = 1 / X, be careful of X = Ø.
c
        WRITE(6,1Ø1)
        CALL TRAPEZ(LOWER, UPPER, TOL, SUM)
        WRITE(6,1Ø4) SUM
        STOP
  1Ø1   FORMAT(/' Trapezoidal integration with end
     1          correction')
  1Ø4   FORMAT(/' Area =', F1Ø.5/)
        END
        SUBROUTINE TRAPEZ(LOWER, UPPER, TOL, SUM)
c
c       Numerical integration by the trapezoidal method.
c
        INTEGER PIECES, I, P2
        REAL X, DELTA, LOWER, UPPER, SUM, TOL
        REAL ENDSUM, MIDSUM, SUM1, ENDCOR
c
        F(X) = 1.Ø / X
        DF(X) = -1.Ø/(X * X)
c
        PIECES = 1
        DELTA = (UPPER - LOWER) / PIECES
        ENDSUM = F(LOWER) + F(UPPER)
        ENDCOR = (DF(UPPER) - DF(LOWER)) / 12.Ø
        SUM = ENDSUM * DELTA / 2.Ø
        WRITE(6,1Ø1) SUM
        MIDSUM = Ø.Ø
  5     PIECES = PIECES * 2
          P2 = PIECES / 2
          SUM1 = SUM
          DELTA = (UPPER - LOWER) / PIECES
          DO 1Ø I = 1, P2
            X = LOWER + DELTA *(2 * I - 1)
            MIDSUM = MIDSUM + F(X)
  1Ø      CONTINUE
```

```
      SUM = (ENDSUM + 2.0*MIDSUM) * DELTA * 0.5 - DELTA
     1          * DELTA * ENDCOR
      WRITE(6,102) PIECES, SUM
      IF (ABS(SUM - SUM1) .GT. ABS(TOL * SUM)) GOTO 5
      RETURN
101   FORMAT(/'        1', F9.5)
102   FORMAT(1X, I7, F9.5)
      END
```

APPENDIX B

SOURCE LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
ADA LINE-BY-LINE TRANSLATION
TELESOFT-ADA COMPILER VERSION 1.5

```
-- LINE BY LINE TRANSLATION OF THE PROGRAM TRAP3.
-- TRAPEZOIDAL METHOD OF INTECRATION.
--
--
   WITH TEXT_IO; USE TEXT_IO;
   USE FLOAT_IO; USE INTEGER_IO;
   PROCEDURE TRAP3 IS

     SUM : FLOAT;
     UPPER : FLOAT := 9.0;
     LOWER : FLOAT := 1.0;
     TOL : FLOAT := 1.0E-5;
--
-- f(X) = 1 / X, be careful of X = 0.
--
   FUNCTION F (X : IN FLOAT) RETURN FLOAT IS
     F : FLOAT;

   BEGIN
     F := 1.0 / X;
     RETURN F;
   END F;

   FUNCTION DF (X : IN FLOAT) RETURN FLOAT IS
     DF : FLOAT;

   BEGIN
     DF := -1.0 /(X*X);
     RETURN DF;
   END DF;

   PROCEDURE TRAPEZ (LOWER,UPPER,TOL,SUM : IN OUT FLOAT) IS
--
-- Numerical integration by the trapezoidal method.
--
     PIECES, I, P2 : INTEGER;
     X, DEL : FLOAT;
     ENDD, ENDSUM, MIDSUM, SUM1, ENDCOR : FLOAT;
--
   BEGIN
     PIECES := 1;
```

```
      DEL := (UPPER - LOWER) / FLOAT(PIECES);
      ENDSUM := F(LOWER) + F(UPPER);
      ENDCOR := (DF(UPPER) - DF(LOWER)) / 12.0;
      SUM := ENDSUM * DEL / 2.0;
      PUT ("       1 "); PUT (SUM);
      NEW_LINE;
      MIDSUM := 0.0;
      <<RETRN>>
      PIECES := PIECES * 2;
      P2 := PIECES / 2;
      SUM1 := SUM;
      DEL := (UPPER - LOWER) / FLOAT(PIECES);
      FOR I IN 1 .. P2 LOOP
        X := LOWER + DEL * FLOAT(2 * I - 1);
        MIDSUM := MIDSUM + F(X);
      END LOOP;
      SUM := (ENDSUM + 2.0*MIDSUM) * DEL * 0.5 - DEL * DEL *
ENDCOR;
      PUT (" "); PUT (PIECES); PUT (" "); PUT (SUM);
      NEW_LINE;
      IF (ABS(SUM-SUM1) > ABS(TOL*SUM)) THEN
        GOTO RETRN;
      END IF;
    END TRAPEZ;
--
--
    BEGIN
      PUT("  Trapezoidal integration with end correction");
NEW_LINE;
      NEW_LINE;
      TRAPEZ (LOWER,UPPER,TOL,SUM);
      NEW_LINE;
      PUT ("  Area ="); PUT (SUM);
      NEW_LINE;
    END TRAP3;
```

APPENDIX C

SOURCE LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
ADA LINE-BY-LINE TRANSLATION USING DEFAULT FLOAT PRECISION
VADS COMPILER RELEASE V04.06

```
-- LINE BY LINE TRANSLATION OF THE PROGRAM TRAP3.
-- TRAPEZOIDAL METHOD OF INTEGRATION.
--
--
   WITH TEXT_IO; USE TEXT_IO;
   PROCEDURE TRAP3 IS

     PACKAGE REAL_IO IS NEW FLOAT_IO(FLOAT);
     PACKAGE INT_IO IS NEW INTEGER_IO(INTEGER);
     USE REAL_IO;
     USE INT_IO;

     SUM : FLOAT := 0.0;
     UPPER : FLOAT := 9.0;
     LOWER : FLOAT := 1.0;
     TOL : FLOAT := 1.0E-5;
--
-- f(X) = 1 / X, be careful of X = 0.
--
   FUNCTION F (X : IN FLOAT) RETURN FLOAT IS
     F : FLOAT;

   BEGIN
     F := 1.0 / X;
     RETURN F;
   END F;

   FUNCTION DF (X : IN FLOAT) RETURN FLOAT IS
     DF : FLOAT;

   BEGIN
     DF := -1.0 /(X*X);
     RETURN DF;
   END DF;

   PROCEDURE TRAPEZ (LOWER,UPPER,TOL,SUM : IN OUT FLOAT) IS
--
-- Numerical integration by the trapezoidal method.
--
     PIECES, P2 : INTEGER;
```

```
      X, DEL : FLOAT;
      ENDD, ENDSUM, MIDSUM, SUM1, ENDCOR : FLOAT;
--
    BEGIN
      PIECES := 1;
      DEL := (UPPER - LOWER) / FLOAT(PIECES);
      ENDSUM := F(LOWER) + F(UPPER);
      ENDCOR := (DF(UPPER) - DF(LOWER)) / 12.0;
      SUM := ENDSUM * DEL / 2.0;
      PUT ("           1 "); PUT (SUM);
      NEW_LINE;
      MIDSUM := 0.0;
      <<RETRN>>
      PIECES := PIECES * 2;
      P2 := PIECES / 2;
      SUM1 := SUM;
      DEL := (UPPER - LOWER) / FLOAT(PIECES);
      FOR I IN 1 .. P2 LOOP
        X := LOWER + DEL * FLOAT(2 * I - 1);
        MIDSUM := MIDSUM + F(X);
      END LOOP;
      SUM := (ENDSUM + 2.0*MIDSUM) * DEL * 0.5 - DEL * DEL *
             ENDCOR;
      PUT (" "); PUT (PIECES); PUT (" "); PUT (SUM);
      NEW_LINE;
      IF (ABS(SUM-SUM1) > ABS(TOL*SUM)) THEN
        GOTO RETRN;
      END IF;
    END TRAPEZ;
--
--
    BEGIN
      PUT("  Trapezoidal integration with end correction");
      NEW_LINE;
      NEW_LINE;
      TRAPEZ (LOWER,UPPER,TOL,SUM);
      NEW_LINE;
      PUT ("  Area ="); PUT (SUM);
      NEW_LINE;
    END TRAP3;
```

APPENDIX D

SOURCE LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
ADA LINE-BY-LINE TRANSLATION USING SIX DIGIT PRECISION
VADS COMPILER RELEASE V04.06

```
-- LINE BY LINE TRANSLATION OF THE PROGRAM TRAP3.
-- TRAPEZOIDAL METHOD OF INTEGRATION, USING SIX
-- DIGIT PRECISION.
--
--
   WITH TEXT_IO; USE TEXT_IO;
   PROCEDURE TRAP3 IS

     type six is digits 6;

     PACKAGE REAL_IO IS NEW FLOAT_IO(six);
     PACKAGE INT_IO IS NEW INTEGER_IO(INTEGER);
     USE REAL_IO;
     USE INT_IO;

     SUM : six := 0.0;
     UPPER : six := 9.0;
     LOWER : six := 1.0;
     TOL : six := 1.0E-5;
--
-- f(X) = 1 / X, be careful of X = 0.
--
   FUNCTION F (X : IN six) RETURN six IS
     F : six;

   BEGIN
     F := 1.0 / X;
     RETURN F;
   END F;

   FUNCTION DF (X : IN six) RETURN six IS
     DF : six;

   BEGIN
     DF := -1.0 /(X*X);
     RETURN DF;
   END DF;

   PROCEDURE TRAPEZ (LOWER,UPPER,TOL,SUM : IN OUT six) IS
--
-- Numerical integration by the trapezoidal method.
--
     PIECES, P2 : INTEGER;
```

```
      X, DEL : six;
      ENDD, ENDSUM, MIDSUM, SUM1, ENDCOR : six;
--
    BEGIN
      PIECES := 1;
      DEL := (UPPER - LOWER) / six(PIECES);
      ENDSUM := F(LOWER) + F(UPPER);
      ENDCOR := (DF(UPPER) - DF(LOWER)) / 12.0;
      SUM := ENDSUM * DEL / 2.0;
      PUT ("            1 "); PUT (SUM);
      NEW_LINE;
      MIDSUM := 0.0;
      <<RETRN>>
      PIECES := PIECES * 2;
      P2 := PIECES / 2;
      SUM1 := SUM;
      DEL := (UPPER - LOWER) / six(PIECES);
      FOR I IN 1 .. P2 LOOP
        X := LOWER + DEL * six(2 * I - 1);
        MIDSUM := MIDSUM + F(X);
      END LOOP;
      SUM := (ENDSUM + 2.0*MIDSUM) * DEL * 0.5 - DEL * DEL *
ENDCOR;
      PUT (" "); PUT (PIECES); PUT (" "); PUT (SUM);
      NEW_LINE;
      IF (ABS(SUM-SUM1) > ABS(TOL*SUM)) THEN
        GOTO RETRN;
      END IF;
    END TRAPEZ;
--
--
    BEGIN
      PUT("  Trapezoidal integration with end correction");
NEW_LINE;
      NEW_LINE;
      TRAPEZ (LOWER,UPPER,TOL,SUM);
      NEW_LINE;
      PUT ("  Area ="); PUT (SUM);
      NEW_LINE;
    END TRAP3;
```

APPENDIX E

SOURCE LISTING
TRAPEZOIDAL INTEGRATION MAIN PROGRAM
ADA REDESIGN USING DEFAULT FLOAT PRECISION
VADS COMPILER RELEASE V04.06

```
with NUMERIC_INTEGRATION; use NUMERIC_INTEGRATION;
with TEXT_IO;             use TEXT_IO;
procedure MAIN is

   package INT_IO  is new INTEGER_IO (INTEGER);
   package REAL_IO is new FLOAT_IO   (FLOAT);
   use INT_IO;
   use REAL_IO;

   UPPER_BOUND : FLOAT := 9.0;
   LOWER_BOUND : FLOAT := 1.0;
   TOLERANCE   : FLOAT := 1.0e-5;
   AREA        : FLOAT;
begin
   NEW_LINE;
   PUT ("TRAPEZOIDAL INTEGRATION");
   NEW_LINE;
   AREA    := (F(UPPER_BOUND)
             + F(LOWER_BOUND))
             *  (UPPER_BOUND - LOWER_BOUND)
             /  2.0;
   PUT (1);
   PUT (AREA);
   TRAPEZOIDAL_INTEGRATION (UPPER_BOUND,
                            LOWER_BOUND,
                            TOLERANCE,
                            AREA);
   NEW_LINE;
   PUT ("AREA = ");
   PUT (AREA);
   NEW_LINE;
end MAIN;
```

APPENDIX F

SOURCE LISTING
TRAPEZOIDAL INTEGRATION ROUTINES PACKAGE
ADA REDESIGN USING DEFAULT FLOAT PRECISION
VADS COMPILER RELEASE V04.06

```
with TEXT_IO; use TEXT_IO;

package NUMERIC_INTEGRATION is

   package INT_IO    is new INTEGER_IO (INTEGER);
   package REAL_IO   is new FLOAT_IO   (FLOAT);
   use INT_IO;
   use REAL_IO;

   procedure TRAPEZOIDAL_INTEGRATION
                              (UPPER_BOUND : in     FLOAT;
                               LOWER_BOUND : in     FLOAT;
                               TOLERANCE   : in     FLOAT;
                               AREA        : in out FLOAT);

   function F  (X : in FLOAT) return FLOAT;
   function DF (X : in FLOAT) return FLOAT;
end NUMERIC_INTEGRATION;
-------------------------------------------------------------
-------------------------------------------------------------
package body NUMERIC_INTEGRATION is

   procedure TRAPEZOIDAL_INTEGRATION
                             UPPER_BOUND : in     FLOAT;
                             LOWER_BOUND : in     FLOAT;
                             TOLERANCE   : in     FLOAT;
                             AREA        : in out FLOAT) is

      NUMBER_OF_PARTITIONS      : INTEGER := 1;
      PREV_NUMBER_OF_PARTITIONS : INTEGER;
      PREVIOUS_AREA             : FLOAT   := 0.0;
      MID_SUM                   : FLOAT   := 0.0;
      END_SUM                   : FLOAT;
      END_CORRECTION            : FLOAT;
      PARTITION_BASE_LENGTH     : FLOAT;
      X                         : FLOAT;

   begin

      END_CORRECTION :=    (DF(UPPER_BOUND)
                          -  DF(LOWER_BOUND))
```

```
                        /  12.0;

      END_SUM          :=     F(UPPER_BOUND)
                           +  F (LOWER_BOUND);

      while ABS (AREA - PREVIOUS_AREA) > ABS(TOLERANCE *
                                              AREA) loop
         PREVIOUS_AREA              :=  AREA;
         PREV_NUMBER_OF_PARTITIONS  :=  NUMBER_OF_PARTITIONS;
         NUMBER_OF_PARTITIONS       :=  NUMBER_OF_PARTITIONS
                                        * 2;
         PARTITION_BASE_LENGTH      :=  (UPPER_BOUND
                                        -  LOWER_BOUND)
                                        /  FLOAT
                                        (NUMBER_OF_PARTITIONS);

         for ITERATION in 1..PREV_NUMBER_OF_PARTITIONS loop
            X          :=  LOWER_BOUND
                           + PARTITION_BASE_LENGTH
                           * FLOAT(2 * ITERATION - 1);
            MID_SUM    :=  MID_SUM + F(X);
         end loop;

         AREA :=  (END_SUM + 2.0 * MID_SUM)
                *  PARTITION_BASE_LENGTH * 0.5
                -  PARTITION_BASE_LENGTH
                *  PARTITION_BASE_LENGTH
                *  END_CORRECTION;
         NEW_LINE;
         PUT (NUMBER_OF_PARTITIONS);
         PUT (AREA);
      end loop;
   end TRAPEZOIDAL_INTEGRATION;
-------------------------------------------------------------
-------------------------------------------------------------
   function F (X : in FLOAT) return FLOAT is
      FUNCTIONAL_VALUE : FLOAT;
   begin
      FUNCTIONAL_VALUE := 1.0 / X;
      return FUNCTIONAL_VALUE;
   end F;
-------------------------------------------------------------
-------------------------------------------------------------
   function DF (X : in FLOAT) return FLOAT is
      FUNCTIONAL_VALUE : FLOAT;
   begin
      FUNCTIONAL_VALUE := -1.0 / (X * X);
      return FUNCTIONAL_VALUE;
   end DF;

end NUMERIC_INTEGRATION;
```

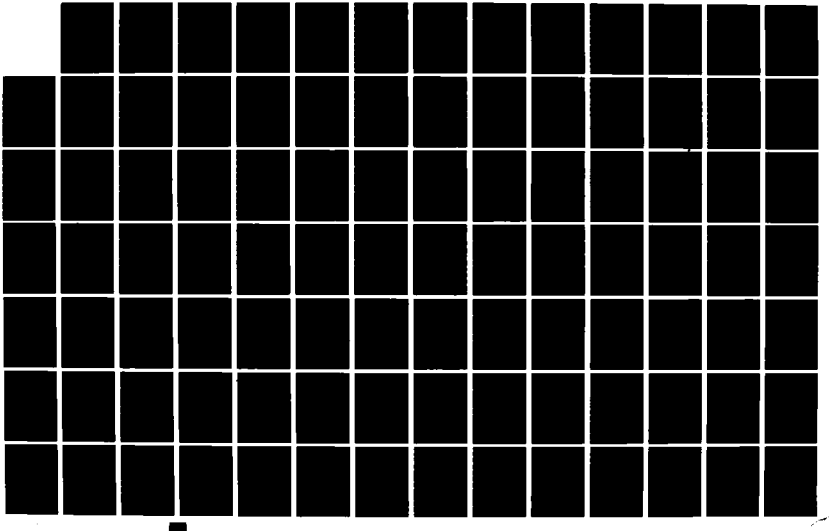
APPENDIX G

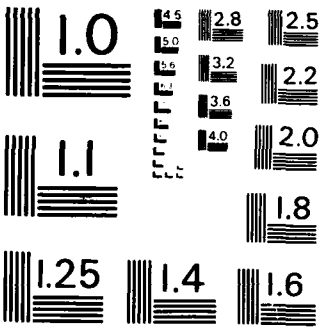
SOURCE LISTING
TRAPEZOIDAL INTEGRATION MAIN PROGRAM
ADA REDESIGN USING SIX DIGIT PRECISION
VADS COMPILER RELEASE V04.06

```
with NUMERIC_INTEGRATION; use NUMERIC_INTEGRATION;
with TEXT_IO;             use TEXT_IO;
procedure MAIN is

   package INT_IO  is new INTEGER_IO (INTEGER);
   package REAL_IO is new FLOAT_IO   (DIGITS_6);
   use INT_IO;
   use REAL_IO;

   UPPER_BOUND : DIGITS_6 := 9.0;
   LOWER_BOUND : DIGITS_6 := 1.0;
   TOLERANCE   : DIGITS_6 := 1.0e-5;
   AREA        : DIGITS_6;
begin
   NEW_LINE;
   PUT ("TRAPEZOIDAL INTEGRATION");
   NEW_LINE;
   AREA    := (F(UPPER_BOUND)
             + F(LOWER_BOUND))
             *  (UPPER_BOUND - LOWER_BOUND)
             /  2.0;
   PUT (1);
   PUT (AREA);
   TRAPEZOIDAL_INTEGRATION (UPPER_BOUND,
                            LOWER_BOUND,
                            TOLERANCE,
                            AREA);
   NEW_LINE;
   PUT ("AREA = ");
   PUT (AREA);
   NEW_LINE;
end MAIN;
```

AD-A161 715 AN ASSESSMENT OF ADA'S SUITABILITY IN GENERAL PURPOSE PROGRAMMING APPLICATIONS(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF SYST.. 2/3

UNCLASSIFIED L D CAVITT ET AL. SEP 85 F/G 9/2 NL





APPENDIX H

SOURCE LISTING
TRAPEZOIDAL INTEGRATION ROUTINES PACKAGE
ADA REDESIGN USING SIX DIGIT PRECISION
VADS COMPILER RELEASE V04.06

```
with TEXT_IO; use TEXT_IO;

package NUMERIC_INTEGRATION is

   type DIGITS_6 is digits 6;

   package INT_IO    is new INTEGER_IO (INTEGER);
   package REAL_IO   is new FLOAT_IO   (DIGITS_6);
   use INT_IO;
   use REAL_IO;

   procedure TRAPEZOIDAL_INTEGRATION
                          (UPPER_BOUND : in     DIGITS_6;
                           LOWER_BOUND : in     DIGITS_6;
                           TOLERANCE   : in     DIGITS_6;
                           AREA        : in out DIGITS_6);

   function F  (X : in DIGITS_6) return DIGITS_6;
   function DF (X : in DIGITS_6) return DIGITS_6;
end NUMERIC_INTEGRATION;
-------------------------------------------------------------
-------------------------------------------------------------
package body NUMERIC_INTEGRATION is

   procedure TRAPEZOIDAL_INTEGRATION
                         (UPPER_BOUND : in     DIGITS_6;
                          LOWER_BOUND : in     DIGITS_6;
                          TOLERANCE   : in     DIGITS_6;
                          AREA        : in out DIGITS_6) is

      NUMBER_OF_PARTITIONS      : INTEGER    := 1;
      PREV_NUMBER_OF_PARTITIONS : INTEGER;
      PREVIOUS_AREA             : DIGITS_6   := 0.0;
      MID_SUM                   : DIGITS_6   := 0.0;
      END_SUM                   : DIGITS_6;
      END_CORRECTION            : DIGITS_6;
      PARTITION_BASE_LENGTH     : DIGITS_6;
      X                         : DIGITS_6;

   begin

      END_CORRECTION :=  (DF(UPPER_BOUND)
```

```
                        -  DF(LOWER_BOUND))
                        /  12.0;
      END_SUM           :=  F(UPPER_BOUND)
                        +  F(LOWER_BOUND);

      while ABS (AREA - PREVIOUS_AREA) > ABS(TOLERANCE *
                                                AREA) loop
         PREVIOUS_AREA              :=  AREA;
         PREV_NUMBER_OF_PARTITIONS  :=  NUMBER_OF_PARTITIONS;
         NUMBER_OF_PARTITIONS       :=  NUMBER_OF_PARTITIONS
                                        * 2;
         PARTITION_BASE_LENGTH      :=  (UPPER_BOUND
                                        -  LOWER_BOUND)
                                        /  DIGITS_6
                                        (NUMBER_OF_PARTITIONS);

         for ITERATION in 1..PREV_NUMBER_OF_PARTITIONS loop
            X          :=  LOWER_BOUND
                           + PARTITION_BASE_LENGTH
                           * DIGITS_6(2 * ITERATION - 1);
            MID_SUM    :=  MID_SUM + F(X);
         end loop;

         AREA :=  (END_SUM + 2.0 * MID_SUM)
                *  PARTITION_BASE_LENGTH * 0.5
                -  PARTITION_BASE_LENGTH
                *  PARTITION_BASE_LENGTH
                *  END_CORRECTION;
         NEW_LINE;
         PUT (NUMBER_OF_PARTITIONS);
         PUT (AREA);
      end loop;
   end TRAPEZOIDAL_INTEGRATION;
-------------------------------------------------------------
-------------------------------------------------------------
   function F (X : in DIGITS_6) return DIGITS_6 is
      FUNCTIONAL_VALUE : DIGITS_6;
   begin
      FUNCTIONAL_VALUE := 1.0 / X;
      return FUNCTIONAL_VALUE;
   end F;
-------------------------------------------------------------
-------------------------------------------------------------
   function DF (X : in DIGITS_6) return DIGITS_6 is
      FUNCTIONAL_VALUE : DIGITS_6;
   begin
      FUNCTIONAL_VALUE := -1.0 / (X * X);
      return FUNCTIONAL_VALUE;
   end DF;

end NUMERIC_INTEGRATION;
```

APPENDIX I

SOURCE LISTING
TRUCK SIMULATION PROGRAM
FORTRAN 4 VERSION WITH 3500 ELEMENT ARRAY

```
C       TRUCK PROBLEM-VARIANT OF PP 77-82 IN BANKS AND CARSON.
        PROGRAM TRUCK
        REAL MIAT,MSVT
        INTEGER NR
        COMMON /SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
       1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
        COMMON /TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
        II=1
        DSEED=567.0
     1  NUMEVS=2
        MIAT= 1.0/3.0
        MSVT=.25
        NCUST=1500
C       WE WILL USE GGUBS TO GENERATE A STRING OF RANDOM #'S
C       ROUTINE GGUBS
        NR=3500
        CALL GGUBS(NR)
C       IIR WILL INDEX THE RANDOM NUMBER GENERATOR.
        IIR=1
C       CALL INITIALIZATION ROUTINE
        CALL INITLZ
C
C
C       CALL TIME-ADVANCE ROUTINE TO DETERMINE IMMINENT EVENT
C       AND ADVANCE
C       CLOCK TO THE IMMINENT EVENT TIME.
    30  CALL TIMADV
C
C       VARIABLE "IMEVT" INDICATES THE IMMINENT EVENT.
C       IMEVT=1 FOR AN ARRIVAL.
C       IMEVT=2 FOR A DEPARTURE.
        GO TO(40,50),IMEVT
    40  CALL ARRVL
        GO TO 30
C       CALL DEPARTURE ROUTINE
    50  CALL DPART
C
C       CHECK TO SEE IF SIMULATION IS OVER. IF NOT RETURLN TO
C       # 30
     8  IF(ND.LT.NCUST) GO TO 30
        IF(II.EQ.1) DSEED=567.0
        IF(II.EQ.2) DSEED=459.0
```

```
      IF(II.EQ.3) DSEED=561.0
      IF(II.EQ.4) DSEED=663.0
      IF(II.EQ.5) DSEED=613.0
      IF(II.EQ.6) DSEED=867.0
      IF(II.EQ.7) DSEED=969.0
      IF(II.EQ.8) DSEED=1071.0
      IF(II.EQ.9) DSEED=1173.0
      IF(II.EQ.10) DSEED=2717.0
      CALL RPTGEN
C     WHEN SIMULATION OVER GENERATE REPORTS.
      II=II+1
      IF(II.EQ.2) DSEED=459.0
      IF(II.EQ.3) DSEED=561.0
      IF(II.EQ.4) DSEED=663.0
      IF(II.EQ.5) DSEED=613.0
      IF(II.EQ.6) DSEED=867.0
      IF(II.EQ.7) DSEED=969.0
      IF(II.EQ.8) DSEED=1071.0
      IF(II.EQ.9) DSEED=1173.0
      IF(II.EQ.10) DSEED=2717.0
   53 IF(II.LE.10) GO TO 1
      STOP
      END
C     INITIALIZATION ROUTINE
      SUBROUTINE INITLZ
      REAL MIAT,MSVT
      COMMON /SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON /TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
C
C     SET SIMULATION CLOCK TO ZERO.
C     ASSUME SYSTEM IS EMPTY AND IDLE AT TIME ZERO.
C     INITIALIZE CUMULATIVE STATISTICS TO 0.
      CLOCK=0.0
      IMEVT=0
      LQT=0
      LST=0
      TLE=0
      B=0
      MQ=0
      S=0
      F=0
      ND=0
C     GENERATE TIME OF FIRST ARRIVAL,IAT, AND SCHEDULE FIRST
C     ARRIVAL
C     IN FEL(1)K.SET FEL(2) TO "INFINITY" TO INDICATE THAT A
C     DEPARTURE
C     IS NOT POSSIBLE WHILE THE SYSTEM IS EMPTY.
      RR=R(IIR)
      X= -log(RR)
      X=X*MIAT
      XXT=1.0
```

```
      FEL(1)=CLOCK + X
      FEL(2)= 1.0E+30
      IIR=IIR+1
      RETURN
      END
C
C
C     TIME ADVANCE ROUTINE: FINDS NEXT EVENT ON FUTURE EVENT
C     LIST AND ADVANCES THE CLOCK.
      SUBROUTINE TIMADV
      REAL MIAT,MSVT
      COMMON /SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON/TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
      FMIN=1.E+29
      IMEVT=0
C     SEARCH FUTURE EVENT LIST FOR NEXT EVENT.
      DO 30 I=1,NUMEVS
      IF(FEL(I).GE.FMIN) GO TO 30
      FMIN=FEL(I)
      IMEVT=I
   30 CONTINUE
      IF(IMEVT.GT.0) GO TO 50
C     ERROR CONDITION: FUTURE EVENT LIST EMPTY.
      WRITE(06,40)
   40 FORMAT(1X,51HFUTURE EVENT LIST EMPTY-SIMULATION CANNOT
     1CONTINUE.)
      CALL RPTGEN
      STOP
C     ADVANCE SUMULATION CLOCK.
C     NEXT EVENT IS TYPE "IMEVT",WHICH WILL OCCUR AT TIME
     1FEL(IMEVT)
C
   50 CLOCK=FEL(IMEVT)
      RETURN
      END
C     ARRIVAL EVENT ROUTINE
      SUBROUTINE ARRVL
      REAL MIAT,MSVT,IAT
      COMMON/SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,LS,F,ND,IIR,R(3500),DSEED
      COMMON/TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
C
C     DETERMINE IF SERVER IS BUSY(IS TRUCK BEING CURRENTLY
C     UNLOADED?)
      IF(LST.EQ.1) GO TO 20
C
C     SERVER IS IDLE. UPDAATE SYSTEM STATE AND RECORD
C     ARRIVAL TIME OF
C     NEW CUSTOMER.
      LST=1
      CHKOUT(1)=CLOCK
C     GENERATE A SERVICE TIME FOR THE NEW ARRIVAL AND
```

```
C       SCHEDULE THE
C       DEPARTURE FOR THIS ARRIVAL.
        RR=R(IIR)
        X=-LOG(RR)
        X=X*MSVT
        FEL(2)=CLOCK+X
        TLE=CLOCK
        IIR=IIR+1
        IF(LQT.GT.MQ) MQ=LQT
        GO TO 100
C
C       SERVER IS BUSY. UP DATE SYSTEM STATE AND RECORD
C       ARRIVAL TIME
C       OF NEW CUSTOMER.
C
   20   LQT=LQT+1
        I=LQT +LST
        IF(I.GT.100) GO TO 200
        CHKOUT(I)=CLOCK
C
C       UPDATE CUMULATIVE STATISTICS B AND MQ. NOTE: S,ND AND
C       F ARE NOT
C       UPDATED WHEN AN ARRIVAL OCCURS.
        B=B+(CLOCK-TLE)
        TLE=CLOCK
        IF(LQT.GT.MQ) MQ=LQT
C
C       GENERATE AN INTER ARRIVAL TIME AND SCHEDULE THE NEXT
C       ARRIVAL
C       EVENT
  100   RR=R(IIR)
        X=-LOG(RR)
        IAT=X*MIAT
        XXT=XXT+1.0
        FEL(1)=CLOCK +IAT
        IIR=IIR+1
        RETURN
C
C       ERROR CONDITION HAS OCCURRED. ARRAY CHKOUT HAS
C       OVERFLOWED.
C       INCREASE DIMENSION OF VARIABLE CHKOUT(I).
  200   WRITE(06,205)
  205   FORMAT(1X,45HOVERFLOW IN ARRAY CHKOUT. INCREASE
       1DIMENSION.,//1X,27HSIMULATION CANNON CONTINUE.)
        CALL RPTGEN
        STOP
        END
C
C       DEPARTURE EVENT ROUTINE.
        SUBROUTINE DPART
        REAL MIAT,MSVT
        COMMON/SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
```

```
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON/TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
C
C     UPDATE CUMULATIVE STATISTICS:B,S,ND,F. NOTE:LQT IS
C     DECREASING
C     SO MQ DOES NOT CHANGE NOW.
      B=B+(CLOCK-TLE)
      TLE=CLOCK
      RT=CLOCK-CHKOUT(1)
      S=S+RT
      ND=ND+1
      IF(RT.GT.4.0) F=F+1
C
C     CHECK CONDITION OF WAITING LINE.
      IF(LQT.GE.1) GO TO 20
C
C     NO CUSTOMES IN LINE. SERVER BECOMES IDLE. NEXT
C     DEPARTURE TIME
C     SET TO "INFINITY".
      LST=0
      FEL(2)=1.E+30
      RETURN
C     AT LEAST ONE CUSTOMES IN LINE, SO MOVE EACH CUSTOMER
C     IN LINE
C     FORWARD ONE SPACE.
   20 DO 30 I=1,LQT
      I1=I+1
      CHKOUT(I)=CHKOUT(I1)
   30 CONTINUE
C     UPDATE SYSTEM STATE
      LQT=LQT-1
C     GENERATE NEW SERVICE TIME FOR CUSTOMER BEGINNING
C     SERVICE,
C     AND SCHEDULE NEXT DEPARTURE EVENT.
      RR=R(IIR)
      X=-LOG(RR)
      SVT=X*MSVT
      FEL(2)=CLOCK +SVT
      IIR=IIR+1
      RETURN
      END
C     REPORT GENERATOR
      SUBROUTINE RPTGEN
      REAL MIAT,MSVT
      COMMON/SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON/TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
C     COMPUTE SUMMARY STATISTICS
      RHO=B/CLOCK
      AVGR=S/ND
      PC4=F/ND
      XX1=S/CLOCK
```

```
      XX2=XXT/CLOCK
      WRITE(06,10)
   10 FORMAT(5X,63HTRUCK QUEUING PROBLEM:ANDERSON AND
     1SWEENEY-SINGLE SERVER QUEUE.,///)
      WRITE(06,15) DSEED,MIAT,MSVT
   15 FORMAT(1X,7HDSEED =,4X,D20.8/1X,25HMEAN ARRIVAL
     1TIME(MIAT) =,4X,F10.4/1X,25HMEAN SERVICE TIME(MSVT)
     2=,4X,F10.4//)
      WRITE(06,30) RHO,MQ,AVGR,PC4,CLOCK,ND,IIR,XX1,S,XX2
   30 FORMAT(1X,38HPROPORTION OF TIME DOCK CREW IS BUSY
     1=,F8.2,//1X,32HMAXIMUM LENGTH OF WAITING LINE
     2=,I8,//1X,28HAVERAGE TIME TO TRANSIT SYS.,F8.2,10H
     3HOURS..//1X,62HPROPORTION OF TRUCKS TAKING FOUR OR
     4MORE HOURS.. IN THE SYSTEM,F6.2//1X,21HSIMULATION RUN
     5LENGTH,F8.2,10H   HOURS..//1X,27HNUMBER OF TRUCKS
     6UNLOADED =,I8//1X,31HNUMBER OF RANDOM NUMBERS USED
     7=,I10,//1X,32HAVERAGE NUMBER OF UNITS IN
     8SYS.=,3X,F8.3//1X,45HTOTAL NUMBER OF TRUCK HOURS IN
     9THE SYSTEM(S)=
     9,F11.3,4X,15H(TRUCKS PER HR)//1X,34HAVERAGE NUMBER OF
     9ARRIVALS PER HR=,4X,F10.4////)
      RETURN
      END
C     RANDOM NUMBER GENERATOR
      SUBROUTINE GGUBS(NR)
      COMMON /SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON /TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
      REAL tmpreal, temp
      INTEGER tmpint
      DO 40 I=1,NR
        tmpreal=DSEED*3.141592
        tmpint=int(tmpreal)
        temp=tmpreal-real(tmpint)
        if (temp .GE. 0.5) then
          tmpint = tmpint + 1
          tmpreal = tmpreal - real(tmpint)
        else
          tmpreal = temp
        endif
        if (tmpreal .LT. 0.0) then
          tmpreal=-tmpreal
        endif
        tmpreal = 2.000*tmpreal
        R(I)=tmpreal
        DSEED=tmpreal
   40 CONTINUE
      RETURN
      END
```

APPENDIX J

SOURCE LISTING
TRUCK SIMULATION PROGRAM
FORTRAN 4 VERSION USING 6500 ELEMENT ARRAY

```
C       TRUCK PROBLEM-VARIANT OF PP 77-82 IN BANKS AND CARSON.
        PROGRAM TRUCK
        REAL MIAT,MSVT
        INTEGER NR
        COMMON /SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
       1CHKOUT(100),B,MQ,S,F,ND,IIR,R(6500),DSEED
        COMMON /TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
        II=1
        DSEED=567.0
     1  NUMEVS=2
        MIAT= 1.0/3.0
        MSVT=.25
        NCUST=3000
C       WE WILL USE GGUBS TO GENERATE A STRING OF RANDOM #'S
C       ROUTINE GGUBS
        NR=3500
        CALL GGUBS(NR)
C       IIR WILL INDEX THE RANDOM NUMBER GENERATOR.
        IIR=1
C       CALL INITIALIZATION ROUTINE
        CALL INITLZ
C
C
C       CALL TIME-ADVANCE ROUTINE TO DETERMINE IMMINENT EVENT
C       AND ADVANCE
C       CLOCK TO THE IMMINENT EVENT TIME.
    30  CALL TIMADV
C
C       VARIABLE "IMEVT" INDICATES THE IMMINENT EVENT.
C       IMEVT=1 FOR AN ARRIVAL.
C       IMEVT=2 FOR A DEPARTURE.
        GO TO(40,50),IMEVT
    40  CALL ARRVL
        GO TO 30
C       CALL DEPARTURE ROUTINE
    50  CALL DPART
C
C       CHECK TO SEE IF SIMULATION IS OVER. IF NOT RETURLN TO
C       # 30
     8  IF(ND.LT.NCUST) GO TO 30
        IF(II.EQ.1) DSEED=567.0
        IF(II.EQ.2) DSEED=459.0
```

```
      IF(II.EQ.3) DSEED=561.0
      IF(II.EQ.4) DSEED=663.0
      IF(II.EQ.5) DSEED=613.0
      IF(II.EQ.6) DSEED=867.0
      IF(II.EQ.7) DSEED=969.0
      IF(II.EQ.8) DSEED=1071.0
      IF(II.EQ.9) DSEED=1173.0
      IF(II.EQ.10) DSEED=2717.0
      CALL RPTGEN
C     WHEN SIMULATION OVER GENERATE REPORTS.
      II=II+1
      IF(II.EQ.2) DSEED=459.0
      IF(II.EQ.3) DSEED=561.0
      IF(II.EQ.4) DSEED=663.0
      IF(II.EQ.5) DSEED=613.0
      IF(II.EQ.6) DSEED=867.0
      IF(II.EQ.7) DSEED=969.0
      IF(II.EQ.8) DSEED=1071.0
      IF(II.EQ.9) DSEED=1173.0
      IF(II.EQ.10) DSEED=2717.0
   53 IF(II.LE.10) GO TO 1
      STOP
      END
C     INITIALIZATION ROUTINE
      SUBROUTINE INITLZ
      REAL MIAT,MSVT
      COMMON /SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON /TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
C
C     SET SIMULATION CLOCK TO ZERO.
C     ASSUME SYSTEM IS EMPTY AND IDLE AT TIME ZERO.
C     INITIALIZE CUMULATIVE STATISTICS TO 0.
      CLOCK=0.0
      IMEVT=0
      LQT=0
      LST=0
      TLE=0
      B=0
      MQ=0
      S=0
      F=0
      ND=0
C     GENERATE TIME OF FIRST ARRIVAL,IAT, AND SCHEDULE FIRST
C     ARRIVAL
C     IN FEL(1)K.SET FEL(2) TO "INFINITY" TO INDICATE THAT A
C     DEPARTURE
C     IS NOT POSSIBLE WHILE THE SYSTEM IS EMPTY.
      RR=R(IIR)
      X= -log(RR)
      X=X*MIAT
      XXT=1.0
```

```
      FEL(1)=CLOCK + X
      FEL(2)= 1.0E+30
      IIR=IIR+1
      RETURN
      END
C
C
C     TIME ADVANCE ROUTINE: FINDS NEXT EVENT ON FUTURE EVENT
C     LIST AND ADVANCES THE CLOCK.
      SUBROUTINE TIMADV
      REAL MIAT,MSVT
      COMMON /SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON/TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
      FMIN=1.E+29
      IMEVT=0
C     SEARCH FUTURE EVENT LIST FOR NEXT EVENT.
      DO 30 I=1,NUMEVS
      IF(FEL(I).GE.FMIN) GO TO 30
      FMIN=FEL(I)
      IMEVT=I
   30 CONTINUE
      IF(IMEVT.GT.0) GO TO 50
C     ERROR CONDITION: FUTURE EVENT LIST EMPTY.
      WRITE(06,40)
   40 FORMAT(1X,51HFUTURE EVENT LIST EMPTY-SIMULATION CANNOT
     1CONTINUE.)
      CALL RPTGEN
      STOP
C     ADVANCE SUMULATION CLOCK.
C     NEXT EVENT IS TYPE "IMEVT",WHICH WILL OCCUR AT TIME
     1FEL(IMEVT)
C
   50 CLOCK=FEL(IMEVT)
      RETURN
      END
C     ARRIVAL EVENT ROUTINE
      SUBROUTINE ARRVL
      REAL MIAT,MSVT,IAT
      COMMON/SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,LS,F,ND,IIR,R(3500),DSEED
      COMMON/TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
C
C     DETERMINE IF SERVER IS BUSY(IS TRUCK BEING CURRENTLY
C     UNLOADED?)
      IF(LST.EQ.1) GO TO 20
C
C     SERVER IS IDLE. UPDAATE SYSTEM STATE AND RECORD
C     ARRIVAL TIME OF
C     NEW CUSTOMER.
      LST=1
      CHKOUT(1)=CLOCK
C     GENERATE A SERVICE TIME FOR THE NEW ARRIVAL AND
```

```
C       SCHEDULE THE
C       DEPARTURE FOR THIS ARRIVAL.
        RR=R(IIR)
        X=-LOG(RR)
        X=X*MSVT
        FEL(2)=CLOCK+X
        TLE=CLOCK
        IIR=IIR+1
        IF(LQT.GT.MQ) MQ=LQT
        GO TO 100
C
C       SERVER IS BUSY. UP DATE SYSTEM STATE AND RECORD
C       ARRIVAL TIME
C       OF NEW CUSTOMER.
C
   20   LQT=LQT+1
        I=LQT +LST
        IF(I.GT.100) GO TO 200
        CHKOUT(I)=CLOCK
C
C       UPDATE CUMULATIVE STATISTICS B AND MQ. NOTE: S,ND AND
C       F ARE NOT
C       UPDATED WHEN AN ARRIVAL OCCURS.
        B=B+(CLOCK-TLE)
        TLE=CLOCK
        IF(LQT.GT.MQ) MQ=LQT
C
C       GENERATE AN INTER ARRIVAL TIME AND SCHEDULE THE NEXT
C       ARRIVAL
C       EVENT
  100   RR=R(IIR)
        X=-LOG(RR)
        IAT=X*MIAT
        XXT=XXT+1.0
        FEL(1)=CLOCK +IAT
        IIR=IIR+1
        RETURN
C
C       ERROR CONDITION HAS OCCURRED. ARRAY CHKOUT HAS
C       OVERFLOWED.
C       INCREASE DIMENSION OF VARIABLE CHKOUT(I).
  200   WRITE(06,205)
  205   FORMAT(1X,45HOVERFLOW IN ARRAY CHKOUT. INCREASE
       1DIMENSION.,//1X,27HSIMULATION CANNON CONTINUE.)
        CALL RPTGEN
        STOP
        END
C
C       DEPARTURE EVENT ROUTINE.
        SUBROUTINE DPART
        REAL MIAT,MSVT
        COMMON/SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
```

```
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON/TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
C
C     UPDATE CUMULATIVE STATISTICS:B,S,ND,F. NOTE:LQT IS
C     DECREASING
C     SO MQ DOES NOT CHANGE NOW.
      B=B+(CLOCK-TLE)
      TLE=CLOCK
      RT=CLOCK-CHKOUT(1)
      S=S+RT
      ND=ND+1
      IF(RT.GT.4.0) F=F+1
C
C     CHECK CONDITION OF WAITING LINE.
      IF(LQT.GE.1) GO TO 20
C
C     NO CUSTOMES IN LINE. SERVER BECOMES IDLE. NEXT
C     DEPARTURE TIME
C     SET TO "INFINITY".
      LST=0
      FEL(2)=1.E+30
      RETURN
C     AT LEAST ONE CUSTOMES IN LINE, SO MOVE EACH CUSTOMER
C     IN LINE
C     FORWARD ONE SPACE.
   20 DO 30 I=1,LQT
      I1=I+1
      CHKOUT(I)=CHKOUT(I1)
   30 CONTINUE
C     UPDATE SYSTEM STATE
      LQT=LQT-1
C     GENERATE NEW SERVICE TIME FOR CUSTOMER BEGINNING
C     SERVICE,
C     AND SCHEDULE NEXT DEPARTURE EVENT.
      RR=R(IIR)
      X=-LOG(RR)
      SVT=X*MSVT
      FEL(2)=CLOCK +SVT
      IIR=IIR+1
      RETURN
      END
C     REPORT GENERATOR
      SUBROUTINE RPTGEN
      REAL MIAT,MSVT
      COMMON/SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON/TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
C     COMPUTE SUMMARY STATISTICS
      RHO=B/CLOCK
      AVGR=S/ND
      PC4=F/ND
      XX1=S/CLOCK
```

```
      XX2=XXT/CLOCK
      WRITE(06,10)
   10 FORMAT(5X,63HTRUCK QUEUING PROBLEM:ANDERSON AND
     1SWEENEY-SINGLE SERVER QUEUE.,///)
      WRITE(06,15) DSEED,MIAT,MSVT
   15 FORMAT(1X,7HDSEED =,4X,D20.8/1X,25HMEAN ARRIVAL
     1TIME(MIAT) =,4X,F10.4/1X,25HMEAN SERVICE TIME(MSVT)
     2=,4X,F10.4//)
      WRITE(06,30) RHO,MQ,AVGR,PC4,CLOCK,ND,IIR,XX1,S,XX2
   30 FORMAT(1X,38HPROPORTION OF TIME DOCK CREW IS BUSY
     1=,F8.2,//1X,32HMAXIMUM LENGTH OF WAITING LINE
     2=,I8,//1X,28HAVERAGE TIME TO TRANSIT SYS.,F8.2,10H
     3HOURS..//1X,62HPROPORTION OF TRUCKS TAKING FOUR OR
     4MORE HOURS.. IN THE SYSTEM,F6.2//1X,21HSIMULATION RUN
     5LENGTH,F8.2,10H   HOURS..//1X,27HNUMBER OF TRUCKS
     6UNLOADED =,I8//1X,31HNUMBER OF RANDOM NUMBERS USED
     7=,I10,//1X,32HAVERAGE NUMBER OF UNITS IN
     8SYS.=,3X,F8.3//1X,45HTOTAL NUMBER OF TRUCK HOURS IN
     9THE SYSTEM(S)=
     9,F11.3,4X,15H(TRUCKS PER HR)//1X,34HAVERAGE NUMBER OF
     9ARRIVALS PER HR=,4X,F10.4////)
      RETURN
      END
C     RANDOM NUMBER GENERATOR
      SUBROUTINE GGUBS(NR)
      COMMON /SIM/ MIAT,MSVT,NCUST,LQT,LST,TLE,
     1CHKOUT(100),B,MQ,S,F,ND,IIR,R(3500),DSEED
      COMMON /TIMEKP/ CLOCK,IMEVT,NUMEVS,FEL(2),XXT
      REAL tmpreal, temp
      INTEGER tmpint
      DO 40 I=1,NR
        tmpreal=DSEED*3.141592
        tmpint=int(tmpreal)
        temp=tmpreal-real(tmpint)
        if (temp .GE. 0.5) then
          tmpint = tmpint + 1
          tmpreal = tmpreal - real(tmpint)
        else
          tmpreal = temp
        endif
        if (tmpreal .LT. 0.0) then
          tmpreal=-tmpreal
        endif
        tmpreal = 2.000*tmpreal
        R(I)=tmpreal
        DSEED=tmpreal
   40 CONTINUE
      RETURN
      END
```

APPENDIX K

SOURCE LISTING
TRUCK SIMULATION PROGRAM
ADA LINE-BY-LINE TRANSLATION WITH 3500 ELEMENT ARRAY
TELESOFT-ADA COMPILER VERSION 1.5

```
------------------------------------------------------------
------------------------------------------------------------
-- THIS PROGRAM IS THE SINGLE SERVER QUEUE SIMULATION
-- PROGRAM WRITTEN IN FORTRAN.  TITLE OF THE PROGRAM IS
-- TRUCK.  THIS PROGRAM IS A LINE BY LINE TRANSLATION OF THE
-- FORTRAN PROGRAM INTO ADA.
------------------------------------------------------------
------------------------------------------------------------

with text_io; use text_io;
use float_io; use integer_io;
with log; use log;

procedure TRUCK is

  NR : integer;
  DSEED : float := 567.0;
  type RN is array(integer range 1 .. 3500) of float;
  type FUTURE_EVENT is array (1 .. 2) of float;
  type ARRIVE is array (1 .. 100) of float;
  MIAT : float := 1.0/3.0;
  MSVT : float := 0.25;
  CLOCK,TLE,B,S : float;
  NUMEVS : integer :=2;
  II : integer := 1;
  LQT,LST,MQ,F : integer;
  ND,IIR,IMEVT : integer;
  XXT : float;
  CHKOUT : ARRIVE;
  R : RN;
  FEL : FUTURE_EVENT;
  NCUST : integer := 1500;

------------------------------------------------------------

  procedure RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED : in out
                                                    float;
                   ND,F,IIR,MQ : in out integer) is
--COMPUTE SUMMARY STATISTICS.
    RHO, AVGR: float;
    XX1, XX2: float;
```

```
  PC4: float;

begin

  RHO := B/CLOCK;
  AVGR := S/float(ND);
  PC4 := float(F)/float(ND);
  XX1 := S/CLOCK;
  XX2 := XXT/CLOCK;
  put("     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-
                                    SINGLE SERVER QUEUE");
  new_line; new_line; new_line ;
  put(" DSEED=    "); put(DSEED); new_line;
  put(" MEAN ARRIVAL TIME(MIAT) =    "); put(MIAT);
  new_line;
  put(" MEAN SERVICE TIME(MSVT) =    "); put(MSVT);
  new_line;
  new_line;
  put(" PROPORTION OF TIME DOCK CREW IS BUSY =");
  put(RHO);
  new_line; new_line;
  put(" MAXIMUM LENGTH OF WAITING LINE ="); put(MQ);
  new_line; new_line;
  put(" AVERAGE TIME TO TRANSIT SYS."); put(AVGR);
  put("HOURS.");
  new_line; new_line;
  put(" PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS..
       IN THE SYSTEM"); put(PC4);
  new_line; new_line;
  put(" SIMULATION RUN LENGTH"); put(CLOCK);
  put("HOURS.");
  new_line; new_line;
  put(" NUMBER OF TRUCKS UNLOADED ="); put(ND);
  new_line; new_line;
  put(" NUMBER OF RANDOM NUMBERS USED ="); put(IIR);
  new_line; new_line;
  put(" AVERAGE NUMBER OF UNITS IN SYS.=   "); put(XX1);
  new_line; new_line;
  put(" TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=");
  put(S);
  put("    (TRUCKS PER HR)");
  new_line; new_line;
  put(" AVERAGE NUMBER OF ARRIVALS PER HR=    ");
  put(XX2);
  new_line; new_line; new_line; new_line;

end RPTGEN;

-------------------------------------------------------------

function GGUBS(DSEED: in float)  return RN is
  tmpint : integer;
```

```
    tmpreal : float;
    SEED : float := DSEED;

  begin

    for I in R'range loop
      tmpreal := SEED*3.141592;
      tmpint := integer(tmpreal);
      tmpreal := tmpreal - float(tmpint);

      if tmpreal < 0.0 then
        tmpreal := -tmpreal;
      end if;

      tmpreal := 2.000*tmpreal;
      R(I) := tmpreal;
      SEED := tmpreal;
    end loop;
    return R;

  end GGUBS;

  ------------------------------------------------------------

  procedure INITLZ(CLOCK,TLE,B,S: in out float;
                   IMEVT,LQT,LST,MQ,F,ND: in out integer;
                   IIR : in out integer;
                   MIAT,XXT : in out float;
                   R : in out RN;
                   FEL : in out FUTURE_EVENT) is
--SET SIMULATION CLOCK TO ZERO.
--ASSUME SYSTEM IS EMPTY AND IDLE AT TIME ZERO.
--INITIALIZE CUMULATIVE STATISTICS TO 0.
    RR: float;
    X: float;
--GENERALTE TIME OF FIRST ARRIVAL, IAT, AND SCHEDULE FIRST
--ARRIVAL IN FEL(1)K.SET FEL(2) TO "INFINITY: TO INDICATE
--THAT A DEPARTURE IS NOT POSSIBLE WHILE THE SYSTEM IS EMPTY

  begin

    CLOCK := 0.0;
    IMEVT := 0;
    LQT := 0;
    LST := 0;
    TLE := 0.0;
    B := 0.0;
    MQ :=0;
    S := 0.0;
    F := 0;
    ND := 0;
    RR := R(IIR);
```

```
    X := -LN(RR);
    X := MIAT * X;
    XXT := 1.0;
    FEL(1) := CLOCK + X;
    FEL(2) := 1.0e30;
    IIR := IIR + 1;

  end INITLZ;

-----------------------------------------------------------------

  procedure TIMADV(IMEVT,NUMEVS,ND,F,IIR,MQ,II : in out
                                                  integer;
                   CLOCK,B,S,XXT,MIAT,MSVT,DSEED : in out
                                                   float;
                   FEL : in out FUTURE_EVENT) is
--TIME ADVANCE ROUTINE: FINDS NEXT EVENT ON
--FUTURE EVENT LIST AND ADVANCES THE CLOCK.
    FMIN: float:= 1.0e29;
--SEARCH FUTURE EVENT LIST FOR NEXT EVENT.

  begin

    IMEVT := 0;
    for I in 1 .. NUMEVS loop

      if FEL(I) >= FMIN then null;

      else

        FMIN := FEL(I);
        IMEVT := I;

      end if;
    end loop;

    if IMEVT > 0 tnen null;

    else
--ERROR CONDITION : FUTURE EVENT LIST EMPTY.
      II := 11;
      PUT(" FUTURE EVENT LIST EMPTY - SIMULATION CANNOT
           CONTINUE.");
      RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED,ND,F,IIR,MQ);
    end if;
--ADVANCE SIMULATION CLOCK
--NEXT EVENT IS TYPE "IMEVT", WHICH WILL OCCUR
--AT TIME FEL(IMEVT).
    CLOCK := FEL(IMEVT);

  end TIMADV;
```

```
------------------------------------------------------------

  procedure ARRVL(LST,LQT,MQ,IIR,ND,F,II : in out integer;
                  CLOCK,B,TLE,MSVT,XXT,MIAT,S,DSEED : in out
                                                      float;
                  CHKOUT : in out ARRIVE;
                  FEL : in out FUTURE_EVENT;
                  R : in out RN) is
--DETERMINE IF SERVER IS BUSY ( IS TRUCK BEING CURRENTLY
--UNLOADED).
    RR,X,IAT : float;
    I : integer;

  begin

    if LST = 1 then
      LQT := LQT +1;
      I := LQT + LST;

      if I > 100 then
--ERROR CONDITION HAS OCCURRED. ARRAY CHKOUT HAS OVERFLOWED.
--INCREASE DIMENSION OF VARIABLE CHKOUT(I).
        II := 11;
        PUT(" OVERFLOW IN ARRAY CHKOUT.  INCREASE
              DIMENSION.,");
        NEW_LINE;
        PUT(" SIMULATION CANNOT CONTINUE.");
        RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED,ND,F,IIR,MQ);

      else

        CHKOUT(I) := CLOCK;
--UPDATE CUMULATIVE STATISTICS B AND MQ. NOTE: S, ND, AND F
--ARE NOT UPDATED WHEN AN ARRIVAL OCCURS.
        B := B + (CLOCK - TLE);
        TLE := CLOCK;

        if LQT > MQ then
          MQ := LQT;
        end if;

--GENERATE AN INTER ARRIVAL TIME AND SCHEDULE THE NEXT
--ARRIVAL EVENT.
        RR := R(IIR);
        X := -LN(RR);
        IAT := MIAT * X;
        XXT := XXT + 1.0;
        FEL(1) := CLOCK + IAT;
        IIR := IIR +1;
      end if;

    else
--SERVER IS IDLE.  UPDATE SYSTEM STATE AND RECORD ARRIVAL T
```

```
--TIME OF NEW CUSTOMER.
      LST := 1;
      CHKOUT(1) := CLOCK;
--GENERATE A SERVICE TIME FOR THE NEW ARRIVAL AND
--SCHEDULE THE DEPARTURE FOR THE ARRIVAL.
      RR:= R(IIR);
      X := -LN(RR);
      X := MSVT * X;
      FEL(2) := CLOCK + X;
      TLE := CLOCK;
      IIR := IIR + 1;

      if LQT > MQ then
        MQ := LQT;
      end if;

      RR := R(IIR);
      X := -LN(RR);
      IAT := MIAT * X;
      XXT := XXT + 1.0;
      FEL(1) := CLOCK + IAT;
      IIR := IIR + 1;
    end if;

  end ARRVL;

-----------------------------------------------------------------

  procedure DPART(B,CLOCK,TLE,S,MSVT : in out float;
                  ND,F,LQT,IIR,LST : in out integer;
                  CHKOUT : in out ARRIVE;
                  R : in out RN;
                  FEL : in out FUTURE_EVENT) is
--UPDATE CUMULATIVE STATISTICS: B, S, ND, F.
--NOTE: LQT IS DECREASING SO MQ DOES NOT CHANGE NOW.
    RT,RR,X,SVT : float;
    I1 : integer;

  begin

    B := B + (CLOCK - TLE);
    TLE := CLOCK;
    RT := CLOCK - CHKOUT(1);
    S := S + RT;
    ND := ND + 1;

    if RT > 4.0 then
      F := F + 1;
    end if;

--CHECK CONDITION OF WAITING LINE.
    if LQT >= 1 then
```

```
      for I in 1 .. LQT loop
        I1 := I + 1;
        CHKOUT(I) := CHKOUT(I1);
      end loop;
--UPDATE SYSTEM STATE.
      LQT := LQT - 1;
--GENERATE NEW SERVICE TIME FOR CUSTOMER BEGINNING
--SERVICE, AND SCHEDULE NEXT DEPARTURE EVENT.
      RR := R(IIR);
      X := -LN(RR);
      SVT := MSVT * X;
      FEL(2) := CLOCK + SVT;
      IIR := IIR + 1;

    else

--NO CUSTOMERS IN LINE.  SERVER BECOMES IDLE.
--NEXT DEPARTURE TIME SET TO "INFINITY".
      LST := 0;
      FEL(2) := 1.0e30;
    end if;

  end DPART;

------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------

  begin

    while II <= 10 loop
--WE WILL USE ONE STRING OF UNIFORM RANDOM NUMBERS
    R := GGUBS(DSEED);
--IIR WILL INDEX THE RANDOM NUMBER GENERATOR.
    IIR := 1;
--CALL INITILIZATION ROUTINE
    INITLZ(CLOCK,TLE,B,S,IMEVT,LQT,LST,MQ,F,ND,IIR,
           MIAT,XXT,R,FEL);
--CALL TIME ADVANCE ROUTINE TO DETERMINE IMMINENT EVENT
--AND ADVANCE CLOCK TO THE IMMINENT EVENT TIME.
     while ND < NCUST loop

TIMADV(IMEVT,NUMEVS,ND,F,IIR,MQ,II,CLOCK,B,S,XXT,MIAT,
              MSVT,DSEED,FEL);
--VARIABLE "IMEVT" INDICATES THE IMMINENT EVENT.
--IMEVT = 1 FOR AN ARRIVAL
--IMEVT = 2 FOR A DEPARTURE
       if IMEVT = 1 then
         ARRVL(LST,LQT,MQ,IIR,ND,F,II,CLOCK,B,TLE,MSVT,XXT,
               MIAT,S,DSEED,CHKOUT,FEL,R);
```

```
        else

DPART(B,CLOCK,TLE,S,MSVT,ND,F,LQT,IIR,LST,CHKOUT,R,FEL);
        end if;

      end loop;
--CHECK TO SEE IF SIMULATION IS OVER. IF NOT RETURN TO
--TIMADV.
     case II is
       when 1 => DSEED := 567.0;
       when 2 => DSEED := 459.0;
       when 3 => DSEED := 561.0;
       when 4 => DSEED := 663.0;
       when 5 => DSEED := 613.0;
       when 6 => DSEED := 867.0;
       when 7 => DSEED := 969.0;
       when 8 => DSEED := 1071.0;
       when 9 => DSEED := 1173.0;
       when 10 => DSEED := 2717.0;
       when others => null;
     end case;

     RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED,ND,F,IIR,MQ);
--WHEN SIMULATION OVER GENERATE REPORTS.
     II := II + 1;

     case II is
       when 2 => DSEED := 459.0;
       when 3 => DSEED := 561.0;
       when 4 => DSEED := 663.0;
       when 5 => DSEED := 613.0;
       when 6 => DSEED := 867.0;
       when 7 => DSEED := 969.0;
       when 8 => DSEED := 1071.0;
       when 9 => DSEED := 1173.0;
       when 10 => DSEED := 2717.0;
       when others => null;
     end case;
     end loop;
     end TRUCK;
```

APPENDIX L

SOURCE LISTING
TRUCK SIMULATION PROGRAM
ADA LINE-BY-LINE TRANSLATION WITH 3500 ELEMENT ARRAY
VADS COMPILER RELEASE V04.06

```
------------------------------------------------------------
------------------------------------------------------------
-- THIS PROGRAM IS THE SINGLE SERVER QUEUE SIMULATION
-- PROGRAM WRITTEN IN FORTRAN.  TITLE OF THE PROGRAM IS
-- TRUCK.  THIS PROGRAM IS A LINE BY LINE TRANSLATION OF THE
-- FORTRAN PROGRAM INTO ADA.
------------------------------------------------------------
------------------------------------------------------------

with text_io; use text_io;
with log; use log;

procedure trk is

  package int_io is new integer_io(integer);
  package real_io is new float_io(float);
  use int_io; use real_io;

  NR : integer;
  DSEED : float := 567.0;
  type RN is array(integer range 1 .. 3500) of float;
  type FUTURE_EVENT is array (1 .. 2) of float;
  type ARRIVE is array (1 .. 100) of float;
  MIAT : float := 1.0/3.0;
  MSVT : float := 0.25;
  CLOCK,TLE,B,S : float;
  NUMEVS : irteger :=2;
  II : integer := 1;
  LQT,LST,MQ,F : integer;
  ND,IIR,IMEVT : integer;
  XXT : float;
  CHKOUT : ARRIVE;
  R : RN;
  FEL : FUTURE_EVENT;
  NCUST : integer := 1500;

------------------------------------------------------------

  procedure RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED : in out
                   float; ND,F,IIR,MQ : in out integer) is
--COMPUTE SUMMARY STATISTICS.
```

```
  RHO, AVGR: float;
  XX1, XX2: float;
  PC4: float;

begin

  RHO := B/CLOCK;
  AVGR := S/float(ND);
  PC4 := float(F)/float(ND);
  XX1 := S/CLOCK;
  XX2 := XXT/CLOCK;
  put("     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-
                                   SINGLE SERVER QUEUE");
  new_line; new_line; new_line;
  put(" DSEED=     "); put(DSEED); new_line;
  put(" MEAN ARRIVAL TIME(MIAT) =     "); put(MIAT);
  new_line;
  put(" MEAN SERVICE TIME(MSVT) =     "); put(MSVT);
  new_line;
  new_line;
  put(" PROPORTION OF TIME DOCK CREW IS BUSY =");
  put(RHO);
  new_line; new_line;
  put(" MAXIMUM LENGTH OF WAITING LINE ="); put(MQ);
  new_line; new_line;
  put(" AVERAGE TIME TO TRANSIT SYS."); put(AVGR);
  put("HOURS.");
  new_line; new_line;
  put(" PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS..
  IN THE SYSTEM"); put(PC4);
  new_line; new_line;
  put(" SIMULATION RUN LENGTH"); put(CLOCK);
  put("HOURS.");
  new_line; new_line;
  put(" NUMBER OF TRUCKS UNLOADED ="); put(ND);
  new_line; new_line;
  put(" NUMBER OF RANDOM NUMBERS USED ="); put(IIR);
  new_line; new_line;
  put(" AVERAGE NUMBER OF UNITS IN SYS.=    "); put(XX1);
  new_line; new_line;
  put(" TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=");
  put(S);
  put("     (TRUCKS PER HR)");
  new_line; new_line;
  put(" AVERAGE NUMBER OF ARRIVALS PER HR=     ");
  put(XX2);
  new_line; new_line; new_line; new_line;

end RPTGEN;

-------------------------------------------------------------
```



```
--CALL TIME ADVANCE ROUTINE TO DETERMINE IMMINENT EVENT
```

```
  function GGUBS(DSEED: in float)  return RN is
    type sixdigit is digits 6;
    tmpint : integer;
    tmpreal, temp : sixdigit;
    SEED : sixdigit;

  begin

    SEED := sixdigit(DSEED);

    for I in R'range loop
      tmpreal := SEED*3.141592;
      tmpint := integer(tmpreal);
      temp := tmpreal - sixdigit(tmpint);

      if temp >= 0.5 then
        tmpint := tmpint + 1;
        tmpreal := tmpreal - sixdigit(tmpint);
      else
        tmpreal := temp;
      end if;

      if tmpreal <= 0.0 then
        tmpreal := -tmpreal;
      end if;

      tmpreal := 2.000*tmpreal;
      R(I) :=float(tmpreal);
      SEED := tmpreal;
    end loop;
    return R;

  end GGUBS;

  ------------------------------------------------------------

  procedure INITLZ(CLOCK,TLE,B,S: in out float;
                   IMEVT,LQT,LST,MQ,F,ND: in out integer;
                   IIR : in out integer;
                   MIAT,XXT : in out float;
                   R : in out RN;
                   FEL : in out FUTURE_EVENT) is
--SET SIMULATION CLOCK TO ZERO.
--ASSUME SYSTEM IS EMPTY AND IDLE AT TIME ZERO.
--INITIALIZE CUMULATIVE STATISTICS TO 0.
    RR: float;
    X: float;
--GENERALTE TIME OF FIRST ARRIVAL, IAT, AND SCHEDULE FIRST
--ARRIVAL IN FEL(1)K.SET FEL(2) TO "INFINITY: TO INDICATE
--THAT A DEPARTURE IS NOT POSSIBLE WHILE THE SYSTEM IS EMPTY

  begin
```

```
      CLOCK := 0.0;
      IMEVT := 0;
      LQT := 0;
      LST := 0;
      TLE := 0.0;
      B := 0.0;
      MQ :=0;
      S := 0.0;
      F := 0;
      ND := 0;
      RR := R(IIR);
      X := -LN(RR);
      X := MIAT * X;
      XXT := 1.0;
      FEL(1) := CLOCK + X;
      FEL(2) := 1.0e30;
      IIR := IIR + 1;

    end INITLZ;

-------------------------------------------------------------

    procedure TIMADV(IMEVT,NUMEVS,ND,F,IIR,MQ,II : in out
                                                    integer;
                     CLOCK,B,S,XXT,MIAT,MSVT,DSEED : in out
                                                    float;
                     FEL : in out FUTURE_EVENT) is
--TIME ADVANCE ROUTINE: FINDS NEXT EVENT ON
--FUTURE EVENT LIST AND ADVANCES THE CLOCK.
      FMIN: float:= 1.0e29;
--SEARCH FUTURE EVENT LIST FOR NEXT EVENT.

    begin

      IMEVT := 0;
      for I in 1 .. NUMEVS loop

        if FEL(I) >= FMIN then null;

        else

          FMIN := FEL(I);
          IMEVT := I;

        end if;
      end loop;

      if IMEVT > 0 then null;

      else
--ERROR CONDITION : FUTURE EVENT LIST EMPTY.
```

```
      II := 11;
      PUT(" FUTURE EVENT LIST EMPTY - SIMULATION CANNOT
                                                CONTINUE.");
      RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED,ND,F,IIR,MQ);
    end if;
--ADVANCE SIMULATION CLOCK
--NEXT EVENT IS TYPE "IMEVT", WHICH WILL OCCUR
--AT TIME FEL(IMEVT).
    CLOCK := FEL(IMEVT);

  end TIMADV;

-------------------------------------------------------------

  procedure ARRVL(LST,LQT,MQ,IIR,ND,F,II : in out integer;
                  CLOCK,B,TLE,MSVT,XXT,MIAT,S,DSEED : in out
                                                      float;
                  CHKOUT : in out ARRIVE;
                  FEL : in out FUTURE_EVENT;
                  R : in out RN) is
--DETERMINE IF SERVER IS BUSY ( IS TRUCK BEING CURRENTLY
--UNLOADED).
    RR,X,IAT : float;
    I : integer;

  begin

    if LST = 1 then
      LQT := LQT +1;
      I := LQT + LST;

      if I > 100 then
--ERROR CONDITION HAS OCCURRED. ARRAY CHKOUT HAS OVERFLOWED.
--INCREASE DIMENSION OF VARIABLE CHKOUT(I).
        II := 11;
        PUT(" OVERFLOW IN ARRAY CHKOUT.  INCREASE
DIMENSION.,");
        NEW_LINE;
        PUT(" SIMULATION CANNOT CONTINUE.");
        RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED,ND,F,IIR,MQ);

      else

        CHKOUT(I) := CLOCK;
--UPDATE CUMULATIVE STATISTICS B AND MQ. NOTE: S, ND, AND F
--ARE NOT UPDATED WHEN AN ARRIVAL OCCURS.
        B := B + (CLOCK - TLE);
        TLE := CLOCK;

        if LQT > MQ then
          MQ := LQT;
        end if;
```

```
--GENERATE AN INTER ARRIVAL TIME AND SCHEDULE THE NEXT
--ARRIVAL EVENT.
        RR := R(IIR);
        X := -LN(RR);
        IAT := MIAT * X;
        XXT := XXT + 1.0;
        FEL(1) := CLOCK + IAT;
        IIR := IIR +1;
      end if;

    else
--SERVER IS IDLE.  UPDATE SYSTEM STATE AND RECORD ARRIVAL
--TIME OF NEW CUSTOMER.
      LST := 1;
      CHKOUT(1) := CLOCK;
--GENERATE A SERVICE TIME FOR THE NEW ARRIVAL AND
--SCHEDULE THE DEPARTURE FOR THE ARRIVAL.
      RR:= R(IIR);
      X := -LN(RR);
      X := MSVT * X;
      FEL(2) := CLOCK + X;
      TLE := CLOCK;
      IIR := IIR + 1;

      if LQT > MQ then
        MQ := LQT;
      end if;

      RR := R(IIR);
      X := -LN(RR);
      IAT := MIAT * X;
      XXT := XXT + 1.0;
      FEL(1) := CLOCK + IAT;
      IIR := IIR + 1;
    end if;

  end ARRVL;

-------------------------------------------------------------

  procedure DPART(B,CLOCK,TLE,S,MSVT : in out float;
                  ND,F,LQT,IIR,LST : in out integer;
                  CHKOUT : in out ARRIVE;
                  R : in out RN;
                  FEL : in out FUTURE_EVENT) is
--UPDATE CUMULATIVE STATISTICS: B, S, ND, F.
--NOTE: LQT IS DECREASING SO MQ DOES NOT CHANGE NOW.
    RT,RR,X,SVT : float;
    I1 : integer;

  begin
```

```
    B := B + (CLOCK - TLE);
    TLE := CLOCK;
    RT := CLOCK - CHKOUT(1);
    S := S + RT;
    ND := ND + 1;

    if RT > 4.0 then
      F := F + 1;
    end if;

--CHECK CONDITION OF WAITING LINE.
    if LQT >= 1 then
      for I in 1 .. LQT loop
        I1 := I + 1;
        CHKOUT(I) := CHKOUT(I1);
      end loop;
--UPDATE SYSTEM STATE.
      LQT := LQT - 1;
--GENERATE NEW SERVICE TIME FOR CUSTOMER BEGINNING
--SERVICE, AND SCHEDULE NEXT DEPARTURE EVENT.
      RR := R(IIR);
      X := -LN(RR);
      SVT := MSVT * X;
      FEL(2) := CLOCK + SVT;
      IIR := IIR + 1;

    else

--NO CUSTOMERS IN LINE.  SERVER BECOMES IDLE.
--NEXT DEPARTURE TIME SET TO "INFINITY".
      LST := 0;
      FEL(2) := 1.0e30;
    end if;

  end DPART;

------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------

  begin

    while II <= 10 loop
--WE WILL USE ONE STRING OF UNIFORM RANDOM NUMBERS
    R := GGUBS(DSEED);
--IIR WILL INDEX THE RANDOM NUMBER GENERATOR.
    IIR := 1;
--CALL INITILIZATION ROUTINE
    INITLZ(CLOCK,TLE,B,S,IMEVT,LQT,LST,MQ,F,ND,IIR,
           MIAT,XXT,R,FEL);
```

```
--CALL TIME ADVANCE ROUTINE TO DETERMINE IMMINENT EVENT
--AND ADVANCE CLOCK TO THE IMMINENT EVENT TIME.
     while ND < NCUST loop

TIMADV(IMEVT,NUMEVS,ND,F,IIR,MQ,II,CLOCK,B,S,XXT,MIAT,
               MSVT,DSEED,FEL);
--VARIABLE "IMEVT" INDICATES THE IMMINENT EVENT.
--IMEVT = 1 FOR AN ARRIVAL
--IMEVT = 2 FOR A DEPARTURE
       if IMEVT = 1 then
         ARRVL(LST,LQT,MQ,IIR,ND,F,II,CLOCK,B,TLE,MSVT,XXT,
               MIAT,S,DSEED,CHKOUT,FEL,R);
       else
DPART(B,CLOCK,TLE,S,MSVT,ND,F,LQT,IIR,LST,CHKOUT,R,FEL);
       end if;
     end loop;
--CHECK TO SEE IF SIMULATION IS OVER. IF NOT RETURN TO
--TIMADV.
    case II is
      when 1 => DSEED := 567.0;
      when 2 => DSEED := 459.0;
      when 3 => DSEED := 561.0;
      when 4 => DSEED := 663.0;
      when 5 => DSEED := 613.0;
      when 6 => DSEED := 867.0;
      when 7 => DSEED := 969.0;
      when 8 => DSEED := 1071.0;
      when 9 => DSEED := 1173.0;
      when 10 => DSEED := 2717.0;
      when others => null;
    end case;

    RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED,ND,F,IIR,MQ);
--WHEN SIMULATION OVER GENERATE REPORTS.
    II := II + 1;

    case II is
      when 2 => DSEED := 459.0;
      when 3 => DSEED := 561.0;
      when 4 => DSEED := 663.0;
      when 5 => DSEED := 613.0;
      when 6 => DSEED := 867.0;
      when 7 => DSEED := 969.0;
      when 8 => DSEED := 1071.0;
      when 9 => DSEED := 1173.0;
      when 10 => DSEED := 2717.0;
      when others => null;
    end case;
    end loop;
    end trk;
```

APPENDIX M

SOURCE LISTING
TRUCK SIMULATION PROGRAM
ADA LINE-BY-LINE TRANSLATION WITH 6500 ELEMENT ARRAY
VADS COMPILER RELEASE V04.06

```
------------------------------------------------------------
------------------------------------------------------------
-- THIS PROGRAM IS THE SINGLE SERVER QUEUE SIMULATION
-- PROGRAM WRITTEN IN FORTRAN.  TITLE OF THE PROGRAM IS
-- TRUCK.  THIS PROGRAM IS A LINE BY LINE TRANSLATION OF THE
-- FORTRAN PROGRAM INTO ADA.
------------------------------------------------------------
------------------------------------------------------------

with text_io; use text_io;
with log; use log;

procedure trk is

  package int_io is new integer_io(integer);
  package real_io is new float_io(float);
  use int_io; use real_io;

  NR : integer;
  DSEED : float := 567.0;
  type RN is array(integer range 1 .. 6500) of float;
  type FUTURE_EVENT is array (1 .. 2) of float;
  type ARRIVE is array (1 .. 100) of float;
  MIAT : float := 1.0/3.0;
  MSVT : float := 0.25;
  CLOCK,TLE,B,S : float;
  NUMEVS : integer :=2;
  II : integer := 1;
  LQT,LST,MQ,F : integer;
  ND,IIR,IMEVT : integer;
  XXT : float;
  CHKOUT : ARRIVE;
  R : RN;
  FEL : FUTURE_EVENT;
  NCUST : integer := 3000;

------------------------------------------------------------

  procedure RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED : in out
                   float; ND,F,IIR,MQ : in out integer) is
--COMPUTE SUMMARY STATISTICS.
```

```
  RHO, AVGR: float;
  XX1, XX2: float;
  PC4: float;

begin

  RHO := B/CLOCK;
  AVGR := S/float(ND);
  PC4 := float(F)/float(ND);
  XX1 := S/CLOCK;
  XX2 := XXT/CLOCK;
  put("     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-
                                   SINGLE SERVER QUEUE");
  new_line; new_line; new_line;
  put(" DSEED=    "); put(DSEED); new_line;
  put(" MEAN ARRIVAL TIME(MIAT) =    "); put(MIAT);
  new_line;
  put(" MEAN SERVICE TIME(MSVT) =    "); put(MSVT);
  new_line;
  new_line;
  put(" PROPORTION OF TIME DOCK CREW IS BUSY =");
  put(RHO);
  new_line; new_line;
  put(" MAXIMUM LENGTH OF WAITING LINE ="); put(MQ);
  new_line; new_line;
  put(" AVERAGE TIME TO TRANSIT SYS."); put(AVGR);
  put("HOURS.");
  new_line; new_line;
  put(" PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS..
  IN THE SYSTEM"); put(PC4);
  new_line; new_line;
  put(" SIMULATION RUN LENGTH"); put(CLOCK);
  put("HOURS.");
  new_line; new_line;
  put(" NUMBER OF TRUCKS UNLOADED ="); put(ND);
  new_line; new_line;
  put(" NUMBER OF RANDOM NUMBERS USED ="); put(IIR);
  new_line; new_line;
  put(" AVERAGE NUMBER OF UNITS IN SYS.=   "); put(XX1);
  new_line; new_line;
  put(" TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=");
  put(S);
  put("    (TRUCKS PER HR)");
  new_line; new_line;
  put(" AVERAGE NUMBER OF ARRIVALS PER HR=    ");
  put(XX2);
  new_line; new_line; new_line; new_line;

end RPTGEN;

-------------------------------------------------------------
```

```
  function GGUBS(DSEED: in float)  return RN is
    type sixdigit is digits 6;
    tmpint : integer;
    tmpreal, temp : sixdigit;
    SEED : sixdigit;

  begin

    SEED := sixdigit(DSEED);

    for I in R'range loop
      tmpreal := SEED*3.141592;
      tmpint := integer(tmpreal);
      temp := tmpreal - sixdigit(tmpint);

      if temp >= 0.5 then
        tmpint := tmpint + 1;
        tmpreal := tmpreal - sixdigit(tmpint);
      else
        tmpreal := temp;
      end if;

      if tmpreal <= 0.0 then
        tmpreal := -tmpreal;
      end if;

      tmpreal := 2.000*tmpreal;
      R(I) :=float(tmpreal);
      SEED := tmpreal;
    end loop;
    return R;

  end GGUBS;

  ------------------------------------------------------------

  procedure INITLZ(CLOCK,TLE,B,S: in out float;
                   IMEVT,LQT,LST,MQ,F,ND: in out integer;
                   IIR : in out integer;
                   MIAT,XXT : in out float;
                   R : in out RN;
                   FEL : in out FUTURE_EVENT) is
--SET SIMULATION CLOCK TO ZERO.
--ASSUME SYSTEM IS EMPTY AND IDLE AT TIME ZERO.
--INITIALIZE CUMULATIVE STATISTICS TO 0.
    RR: float;
    X: float;
--GENERALTE TIME OF FIRST ARRIVAL, IAT, AND SCHEDULE FIRST
--ARRIVAL IN FEL(1)K.SET FEL(2) TO "INFINITY: TO INDICATE
--THAT A DEPARTURE IS NOT POSSIBLE WHILE THE SYSTEM IS EMPTY

  begin
```

```
    CLOCK := 0.0;
    IMEVT := 0;
    LQT := 0;
    LST := 0;
    TLE := 0.0;
    B := 0.0;
    MQ :=0;
    S := 0.0;
    F := 0;
    ND := 0;
    RR := R(IIR);
    X := -LN(RR);
    X := MIAT * X;
    XXT := 1.0;
    FEL(1) := CLOCK + X;
    FEL(2) := 1.0e30;
    IIR := IIR + 1;

  end INITLZ;

-------------------------------------------------------------

  procedure TIMADV(IMEVT,NUMEVS,ND,F,IIR,MQ,II : in out
                                                  integer;
                   CLOCK,B,S,XXT,MIAT,MSVT,DSEED : in out
                                                  float;
                   FEL : in out FUTURE_EVENT) is
--TIME ADVANCE ROUTINE: FINDS NEXT EVENT ON
--FUTURE EVENT LIST AND ADVANCES THE CLOCK.
    FMIN: float:= 1.0e29;
--SEARCH FUTURE EVENT LIST FOR NEXT EVENT.

  begin

    IMEVT := 0;
    for I in 1 .. NUMEVS loop

      if FEL(I) >= FMIN then null;

      else

        FMIN := FEL(I);
        IMEVT := I;

      end if;
    end loop;

    if IMEVT > 0 then null;

    else
--ERROR CONDITION : FUTURE EVENT LIST EMPTY.
```

```
      II := 11;
      PUT(" FUTURE EVENT LIST EMPTY - SIMULATION CANNOT
                                                CONTINUE.");
      RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED,ND,F,IIR,MQ);
    end if;
--ADVANCE SIMULATION CLOCK
--NEXT EVENT IS TYPE "IMEVT", WHICH WILL OCCUR
--AT TIME FEL(IMEVT).
    CLOCK := FEL(IMEVT);

  end TIMADV;

-------------------------------------------------------------

  procedure ARRVL(LST,LQT,MQ,IIR,ND,F,II : in out integer;
                  CLOCK,B,TLE,MSVT,XXT,MIAT,S,DSEED : in out
                                                      float;
                  CHKOUT : in out ARRIVE;
                  FEL : in out FUTURE_EVENT;
                  R : in out RN) is
--DETERMINE IF SERVER IS BUSY ( IS TRUCK BEING CURRENTLY
--UNLOADED).
    RR,X,IAT : float;
    I : integer;

  begin

    if LST = 1 then
      LQT := LQT +1;
      I := LQT + LST;

      if I > 100 then
--ERROR CONDITION HAS OCCURRED. ARRAY CHKOUT HAS OVERFLOWED.
--INCREASE DIMENSION OF VARIABLE CHKOUT(I).
        II := 11;
        PUT(" OVERFLOW IN ARRAY CHKOUT.  INCREASE
DIMENSION.,");
        NEW_LINE;
        PUT(" SIMULATION CANNOT CONTINUE.");
        RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED,ND,F,IIR,MQ);

      else

        CHKOUT(I) := CLOCK;
--UPDATE CUMULATIVE STATISTICS B AND MQ. NOTE: S, ND, AND F
--ARE NOT UPDATED WHEN AN ARRIVAL OCCURS.
        B := B + (CLOCK - TLE);
        TLE := CLOCK;

        if LQT > MQ then
          MQ := LQT;
        end if;
```

```
--GENERATE AN INTER ARRIVAL TIME AND SCHEDULE THE NEXT
--ARRIVAL EVENT.
        RR := R(IIR);
        X := -LN(RR);
        IAT := MIAT * X;
        XXT := XXT + 1.0;
        FEL(1) := CLOCK + IAT;
        IIR := IIR +1;
      end if;

    else
--SERVER IS IDLE.  UPDATE SYSTEM STATE AND RECORD ARRIVAL
--TIME OF NEW CUSTOMER.
      LST := 1;
      CHKOUT(1) := CLOCK;
--GENERATE A SERVICE TIME FOR THE NEW ARRIVAL AND
--SCHEDULE THE DEPARTURE FOR THE ARRIVAL.
      RR:= R(IIR);
      X := -LN(RR);
      X := MSVT * X;
      FEL(2) := CLOCK + X;
      TLE := CLOCK;
      IIR := IIR + 1;

      if LQT > MQ then
        MQ := LQT;
      end if;

      RR := R(IIR);
      X := -LN(RR);
      IAT := MIAT * X;
      XXT := XXT + 1.0;
      FEL(1) := CLOCK + IAT;
      IIR := IIR + 1;
    end if;

  end ARRVL;

-------------------------------------------------------------

  procedure DPART(B,CLOCK,TLE,S,MSVT : in out float;
                  ND,F,LQT,IIR,LST : in out integer;
                  CHKOUT : in out ARRIVE;
                  R : in out RN;
                  FEL : in out FUTURE_EVENT) is
--UPDATE CUMULATIVE STATISTICS: B, S, ND, F.
--NOTE: LQT IS DECREASING SO MQ DOES NOT CHANGE NOW.
    RT,RR,X,SVT : float;
    I1 : integer;

  begin
```

```
    B := B + (CLOCK - TLE);
    TLE := CLOCK;
    RT := CLOCK - CHKOUT(1);
    S := S + RT;
    ND := ND + 1;

    if RT > 4.0 then
      F := F + 1;
    end if;

--CHECK CONDITION OF WAITING LINE.
    if LQT >= 1 then
      for I in 1 .. LQT loop
        I1 := I + 1;
        CHKOUT(I) := CHKOUT(I1);
      end loop;
--UPDATE SYSTEM STATE.
      LQT := LQT - 1;
--GENERATE NEW SERVICE TIME FOR CUSTOMER BEGINNING
--SERVICE, AND SCHEDULE NEXT DEPARTURE EVENT.
      RR := R(IIR);
      X := -LN(RR);
      SVT := MSVT * X;
      FEL(2) := CLOCK + SVT;
      IIR := IIR + 1;

    else

--NO CUSTOMERS IN LINE.  SERVER BECOMES IDLE.
--NEXT DEPARTURE TIME SET TO "INFINITY".
      LST := 0;
      FEL(2) := 1.0e30;
    end if;

  end DPART;

-------------------------------------------------------------
-------------------------------------------------------------
-------------------------------------------------------------
-------------------------------------------------------------

  begin

    while II <= 10 loop
--WE WILL USE ONE STRING OF UNIFORM RANDOM NUMBERS
    R := GGUBS(DSEED);
--IIR WILL INDEX THE RANDOM NUMBER GENERATOR.
    IIR := 1;
--CALL INITILIZATION ROUTINE
    INITL2(CLOCK,TLE,B,S,IMEVT,LQT,LST,MQ,F,ND,IIR,
           MIAT,XXT,R,FEL);
```

```
--CALL TIME ADVANCE ROUTINE TO DETERMINE IMMINENT EVENT
--AND ADVANCE CLOCK TO THE IMMINENT EVENT TIME.
     while ND < NCUST loop

TIMADV(IMEVT,NUMEVS,ND,F,IIR,MQ,II,CLOCK,B,S,XXT,MIAT,
               MSVT,DSEED,FEL);
--VARIABLE "IMEVT" INDICATES THE IMMINENT EVENT.
--IMEVT = 1 FOR AN ARRIVAL
--IMEVT = 2 FOR A DEPARTURE
       if IMEVT = 1 then
         ARRVL(LST,LQT,MQ,IIR,ND,F,II,CLOCK,B,TLE,MSVT,XXT,
               MIAT,S,DSEED,CHKOUT,FEL,R);
       else
DPART(B,CLOCK,TLE,S,MSVT,ND,F,LQT,IIR,LST,CHKOUT,R,FEL);
       end if;
     end loop;
--CHECK TO SEE IF SIMULATION IS OVER. IF NOT RETURN TO
--TIMADV.
    case II is
      when 1 => DSEED := 567.0;
      when 2 => DSEED := 459.0;
      when 3 => DSEED := 561.0;
      when 4 => DSEED := 663.0;
      when 5 => DSEED := 613.0;
      when 6 => DSEED := 867.0;
      when 7 => DSEED := 969.0;
      when 8 => DSEED := 1071.0;
      when 9 => DSEED := 1173.0;
      when 10 => DSEED := 2717.0;
      when others => null;
    end case;

    RPTGEN(B,CLOCK,S,XXT,MIAT,MSVT,DSEED,ND,F,IIR,MQ);
--WHEN SIMULATION OVER GENERATE REPORTS.
    II := II + 1;

    case II is
      when 2 => DSEED := 459.0;
      when 3 => DSEED := 561.0;
      when 4 => DSEED := 663.0;
      when 5 => DSEED := 613.0;
      when 6 => DSEED := 867.0;
      when 7 => DSEED := 969.0;
      when 8 => DSEED := 1071.0;
      when 9 => DSEED := 1173.0;
      when 10 => DSEED := 2717.0;
      when others => null;
    end case;
    end loop;
    end trk;
```

APPENDIX N

SOURCE LISTING
TRUCK SIMULATION MAIN PROGRAM
ADA REDESIGN
TELESOFT-ADA VERSION 1.5

```
with TEXT_IO               ; use TEXT_IO;
                             use FLOAT_IO;
with SIMULATION_ROUTINES ; use SIMULATION_ROUTINES;

procedure TRUCK_SIMULATION is
   MEAN_INTER_ARRIVAL_TIME : FLOAT        := 1.0/3.0;
   MEAN_SERVICE_TIME       : FLOAT        := 0.25  ;
   STATS                   : STATISTICS;
   SERVICE_QUEUE           : QUEUE;
   RANDOM_NUMBER           : RANDOM_NUMBER_RECORD;
begin
   while STATS.REPETITION < 10 loop
      INITIALIZE (STATS,
                  SERVICE_QUEUE,
                  MEAN_INTER_ARRIVAL_TIME,
                  RANDOM_NUMBER);

      while STATS.TOTAL_DEPARTURES < 1500 loop
         if STATS.NEXT_ARRIVAL < STATS.NEXT_DEPARTURE then
            GENERATE_ARRIVAL (STATS,
                              SERVICE_QUEUE,
                              MEAN_INTER_ARRIVAL_TIME,
                              MEAN_SERVICE_TIME,
                              RANDOM_NUMBER);
         else
            GENERATE_DEPARTURE (STATS,
                                SERVICE_QUEUE,
                                MEAN_SERVICE_TIME,
                                RANDOM_NUMBER);
         end if;
      end loop;
      GENERATE_REPORT (STATS,
                       SERVICE_QUEUE,
                       MEAN_INTER_ARRIVAL_TIME,
                       MEAN_SERVICE_TIME,
                       RANDOM_NUMBER);
   end loop;
end TRUCK_SIMULATION;
```

APPENDIX O

SOURCE LISTING
SIMULATION ROUTINES PACKAGE
ADA REDESIGN
TELESOFT-ADA VERSION 1.5

```
with TEXT_IO; use TEXT_IO;
              use FLOAT_IO;
              use INTEGER_IO;
with LOG;     use LOG;

package SIMULATION_ROUTINES is

   type SEED_ARRAY is array (INTEGER range 1..10 ) of FLOAT;

   type RANDOM_NUMBER_RECORD is record
        NUMBER  :   FLOAT;
        SEEDS   :   SEED_ARRAY := ( 1 =>  567.0,
                                    2 =>  459.0,
                                    3 =>  561.0,
                                    4 =>  663.0,
                                    5 =>  613.0,
                                    6 =>  867.0,
                                    7 =>  969.0,
                                    8 => 1071.0,
                                    9 => 1173.0,
                                   10 => 2717.0);
        DSEED   :   FLOAT;
        COUNT   :   INTEGER;
   end record;

   type SIMPLE_ARRAY is
                       array (INTEGER range 1..100) of FLOAT;

   type STATISTICS is record
        CLOCK                   : FLOAT;
        NEXT_ARRIVAL            : FLOAT;
        NEXT_DEPARTURE          : FLOAT;
        TIME_LAST_EVENT         : FLOAT;
        SERVER_BUSY_TIME        : FLOAT;
        TOTAL_TIME_IN_SYSTEM    : FLOAT;
        TOTAL_ARRIVALS          : INTEGER;
        TOTAL_DEPARTURES        : INTEGER;
        MAX_Q_LENGTH            : INTEGER;
        FOUR_HOURS_IN_SYSTEM    : INTEGER;
        REPETITION              : INTEGER := 0;
   end record;
```

```
   type QUEUE is record
        ELEMENT : SIMPLE_ARRAY;
        LENGTH  : INTEGER := 0;
        IS_IDLE : BOOLEAN := TRUE;
   end record;

   procedure GENERATE_ARRIVAL
             (STATS                     : in out STATISTICS;
              SERVICE_QUEUE             : in out QUEUE;
              MEAN_INTER_ARRIVAL_TIME   : in     FLOAT;
              MEAN_SERVICE_TIME         : in     FLOAT;
              RANDOM_NUMBER             : in out
                                          RANDOM_NUMBER_RECORD);

   procedure GENERATE_DEPARTURE
             (STATS              : in out STATISTICS;
              SERVICE_QUEUE      : in out QUEUE;
              MEAN_SERVICE_TIME  : in     FLOAT;
              RANDOM_NUMBER      : in out
                                   RANDOM_NUMBER_RECORD);

   procedure INITIALIZE
             (STATS                     : in out STATISTICS;
              SERVICE_QUEUE             : in out QUEUE;
              MEAN_INTER_ARRIVAL_TIME   : in     FLOAT;
              RANDOM_NUMBER             : in out
                                          RANDOM_NUMBER_RECORD);

   procedure GENERATE_REPORT
             (STATS                     : in out STATISTICS;
              SERVICE_QUEUE             : in out QUEUE;
              MEAN_INTER_ARRIVAL_TIME   : in     FLOAT;
              MEAN_SERVICE_TIME         : in     FLOAT;
              RANDOM_NUMBER             : in out
                                          RANDOM_NUMBER_RECORD);

   procedure RAN
             (RANDOM_NUMBER :  in out RANDOM_NUMBER_RECORD);

end SIMULATION_ROUTINES;
-------------------------------------------------------------
-------------------------------------------------------------
package body SIMULATION_ROUTINES is

   procedure GENERATE_ARRIVAL
             (STATS                     : in out STATISTICS;
              SERVICE_QUEUE             : in out QUEUE;
              MEAN_INTER_ARRIVAL_TIME   : in     FLOAT;
              MEAN_SERVICE_TIME         : in     FLOAT;
              RANDOM_NUMBER             : in out
                                        RANDOM_NUMBER_RECORD) is
```

```
   SERVICE_TIME        : FLOAT;
   INTER_ARRIVAL_TIME  : FLOAT;

begin
   STATS.CLOCK           := STATS.NEXT_ARRIVAL;
   SERVICE_QUEUE.LENGTH  := SERVICE_QUEUE.LENGTH + 1;
   SERVICE_QUEUE.ELEMENT (SERVICE_QUEUE.LENGTH)
                         := STATS.CLOCK;
   if SERVICE_QUEUE.IS_IDLE then
      SERVICE_QUEUE.IS_IDLE := FALSE;
      RAN (RANDOM_NUMBER);
      SERVICE_TIME           :=  MEAN_SERVICE_TIME
                               * (-LN
                                 (RANDOM_NUMBER.NUMBER));
      STATS.NEXT_DEPARTURE   :=  STATS.CLOCK
                               + SERVICE_TIME;
   else
      STATS.SERVER_BUSY_TIME :=   STATS.SERVER_BUSY_TIME
                                + (STATS.CLOCK
                                -  STATS.TIME_LAST_EVENT);
   end if;
   STATS.TIME_LAST_EVENT := STATS.CLOCK;
   STATS.TOTAL_ARRIVALS  := STATS.TOTAL_ARRIVALS + 1;
   if SERVICE_QUEUE.LENGTH > STATS.MAX_Q_LENGTH then
      STATS.MAX_Q_LENGTH := SERVICE_QUEUE.LENGTH;
   end if;
   RAN (RANDOM_NUMBER);
   INTER_ARRIVAL_TIME :=    MEAN_INTER_ARRIVAL_TIME
                          *  (-LN (RANDOM_NUMBER.NUMBER));
   STATS.NEXT_ARRIVAL :=    STATS.CLOCK
                          +  INTER_ARRIVAL_TIME;
end GENERATE_ARRIVAL;
-------------------------------------------------------------
-------------------------------------------------------------
procedure GENERATE_DEPARTURE
                 (STATS               : in out STATISTICS;
                  SERVICE_QUEUE       : in out QUEUE;
                  MEAN_SERVICE_TIME   : in     FLOAT;
                  RANDOM_NUMBER       : in out
                                        RANDOM_NUMBER_RECORD) is

   TIME_IN_SYSTEM_THIS_DEPARTURE : FLOAT;
   SERVICE_TIME                  : FLOAT;

begin
   STATS.CLOCK                 :=  STATS.NEXT_DEPARTURE;
   STATS.SERVER_BUSY_TIME      :=  STATS.SERVER_BUSY_TIME
                                 + (STATS.CLOCK
                                 - STATS.TIME_LAST_EVENT);
   STATS.TIME_LAST_EVENT          :=   STATS.CLOCK;
   TIME_IN_SYSTEM_THIS_DEPARTURE :=   STATS.CLOCK
                                 - SERVICE_QUEUE.ELEMENT(1);
   STATS.TOTAL_TIME_IN_SYSTEM
```

```
                        :=   STATS.TOTAL_TIME_IN_SYSTEM
                           + TIME_IN_SYSTEM_THIS_DEPARTURE;
   STATS.TOTAL_DEPARTURES     :=   STATS.TOTAL_DEPARTURES
                              + 1;

   if TIME_IN_SYSTEM_THIS_DEPARTURE > 4.0 then
      STATS.FOUR_HOURS_IN_SYSTEM
                        := STATS.FOUR_HOURS_IN_SYSTEM + 1;
   end if;

   if SERVICE_QUEUE.LENGTH - 1 = 0 then    -- if queue
                                           -- will be
                                           -- empty
                                           -- after this
                                           -- departure
      SERVICE_QUEUE.LENGTH  := 0;
      SERVICE_QUEUE.IS_IDLE := TRUE;
      STATS.NEXT_DEPAPTURE  := 1.0e30;
   else
      for INDEX in 1..SERVICE_QUEUE.LENGTH - 1 loop
         SERVICE_QUEUE.ELEMENT (INDEX)
                        := SERVICE_QUEUE.ELEMENT(INDEX + 1);
      end loop;
      SERVICE_QUEUE.LENGTH  := SERVICE_QUEUE.LENGTH - 1;
      RAN (RANDOM_NUMBER);
      SERVICE_TIME          :=  MEAN_SERVICE_TIME
                              * (-LN
                                 (RANDOM_NUMBER.NUMBER));
      STATS.NEXT_DEPARTURE  :=  STATS.CLOCK
                              + SERVICE_TIME;
   end if;
end GENERATE_DEPARTURE;
-------------------------------------------------------------
-------------------------------------------------------------
procedure INITIALIZE
             (STATS                      : in out
                                           STATISTICS;
              SERVICE_QUEUE              : in out QUEUE;
              MEAN_INTER_ARRIVAL_TIME    : in     FLOAT;
              RANDOM_NUMBER              : in out
                                 RANDOM_NUMBER_RECORD) is
   ARRIVAL_TIME  : FLOAT;
begin
   STATS.REPETITION     := STATS.REPETITION + 1;
   RANDOM_NUMBER.DSEED := RANDOM_NUMBER.SEEDS
                          (STATS.REPETITION);
   RANDOM_NUMBER.COUNT := 0;
   RAN (RANDOM_NUMBER);
   ARRIVAL_TIME          :=  MEAN_INTER_ARRIVAL_TIME
                           * (-LN (RANDOM_NUMBER.NUMBER));
   STATS.CLOCK                := 0.0;
   STATS.TIME_LAST_EVENT      := 0.0;
```

```
   STATS.SERVER_BUSY_TIME      := 0.0;
   STATS.TOTAL_TIME_IN_SYSTEM  := 0.0;
   STATS.TOTAL_ARRIVALS        := 0  ;
   STATS.TOTAL_DEPARTURES      := 0  ;
   STATS.MAX_Q_LENGTH          := 0  ;
   STATS.FOUR_HOURS_IN_SYSTEM  := 0  ;
   STATS.NEXT_ARRIVAL          :=  STATS.CLOCK
                                 + ARRIVAL_TIME;
   STATS.NEXT_DEPARTURE        := 1.0e30;
   SERVICE_QUEUE.LENGTH        := 0;
   SERVICE_QUEUE.IS_IDLE       := TRUE;
end INITIALIZE;
-------------------------------------------------------------
-------------------------------------------------------------
procedure GENERATE_REPORT
                (STATS                        : in out
                                                STATISTICS;
                 SERVICE_QUEUE                : in out QUEUE;
                 MEAN_INTER_ARRIVAL_TIME      : in     FLOAT;
                 MEAN_SERVICE_TIME            : in     FLOAT;
                 RANDOM_NUMBER                : in out
                                  RANDOM_NUMBER_RECORD) is
   TEMP : FLOAT;
begin
   NEW_LINE;
   NEW_LINE;
   PUT ("RANDOM NUMBER GENERATOR SEED");
   PUT (RANDOM_NUMBER.SEEDS (STATS.REPETITION));
   NEW_LINE;
   PUT ("MEAN INTERARRIVAL TIME = ");
   PUT (MEAN_INTER_ARRIVAL_TIME);
   NEW_LINE;
   PUT ("MEAN SERVICE TIME = ");
   PUT (MEAN_SERVICE_TIME);
   NEW_LINE;
   NEW_LINE;
   PUT ("PROPORTION OF TIME DOCK CREW IS BUSY = ");
   TEMP := STATS.SERVER_BUSY_TIME / STATS.CLOCK;
   PUT (TEMP);
   NEW_LINE;
   NEW_LINE;
   PUT ("MAXIMUM LENGTH OF WAITING LINE = ");
   PUT (STATS.MAX_Q_LENGTH);
   NEW_LINE;
   NEW_LINE;
   PUT ("AVERAGE TIME TO TRANSIT SYSTEM = ");
   TEMP :=          STATS.TOTAL_TIME_IN_SYSTEM
          / FLOAT(STATS.TOTAL_DEPARTURES);
   PUT (TEMP);
   NEW_LINE;
   NEW_LINE;
   PUT ("PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS
```

```
        ="");
      TEMP :=     FLOAT (STATS.FOUR_HOURS_IN_SYSTEM)
              /   FLOAT (STATS.TOTAL_DEPARTURES);
      PUT (TEMP);
      NEW_LINE;
      NEW_LINE;
      PUT ("SIMULATION RUN LENGTH = ");
      PUT (STATS.CLOCK);
      PUT ("  HOURS");
      NEW_LINE;
      NEW_LINE;
      PUT ("NUMBER OF TRUCKS UNLOADED = ");
      PUT (STATS.TOTAL_DEPARTURES);
      NEW_LINE;
      NEW_LINE;
      PUT ("NUMBER OF RANDOM NUMBERS USED = ");
      PUT (RANDOM_NUMBER.COUNT);
      NEW_LINE;
      NEW_LINE;
      PUT ("AVERAGE NUMBER OF UNITS IN SYS. = ");
      TEMP := STATS.TOTAL_TIME_IN_SYSTEM / STATS.CLOCK;
      PUT (TE  );
      NEW_LINE;
      NEW_LINE;
      PUT ("TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
           ");
      PUT (STATS.TOTAL_TIME_IN_SYSTEM);
      NEW_LINE;
      NEW_LINE;
      PUT ("AVERAGE NUMBER OF ARRIVALS PER HR = ");
      TEMP := FLOAT (STATS.TOTAL_ARRIVALS) / STATS.CLOCK;
      PUT (TEMP);
      NEW_LINE;
      NEW_LINE;
   end GENERATE_REPORT;
-------------------------------------------------------------
  procedure RAN (RANDOM_NUMBER :  in out
                                    RANDOM_NUMBER_RECORD) is
     tmpint : integer;
     tmpreal : float;
  begin
     tmpreal := RANDOM_NUMBER.DSEED*3.141592;
     tmpint := integer(tmpreal);
     tmpreal := tmpreal - float(tmpint);
     if tmpreal < 0.0 then
       tmpreal := -tmpreal;
     end if;
     tmpreal := 2.0 * tmpreal;
     RANDOM_NUMBER.NUMBER := tmpreal;
     RANDOM_NUMBER.DSEED  := tmpreal;
     RANDOM_NUMBER.COUNT  := RANDOM_NUMBER.COUNT + 1;
  end RAN;
end SIMULATION_ROUTINES;
```

APPENDIX P

SOURCE LISTING
NATURAL LOG PACKAGE USED BY
ADA TRUCK SIMULATION PROGRAM

```
------------------------------------------------------------
------------------------------------------------------------
--   CALCULATES NATURAL LOGS, EXPONENTIATION AND SQRTS
------------------------------------------------------------
------------------------------------------------------------

package log is
  function LN (x: in float) return float;
  function LN (x: in integer) return float;
  function "**" (a: in float; x: in float) return float;
  function SQRT (a: in float) return float;

  bounds_error : exception;
end log;
------------------------------------------------------------
------------------------------------------------------------
package body log is

  function LN (x: in float) return float is
    result : float;
    old: float;
    term: float;
    power: float;
  begin
    if x>0.0 then
      old :=0.0;
      term :=(x-1.0)/(x+1.0);
      result :=2.0*term;
      power :=term;
      for index in 1 .. integer'last loop
        power :=power*term*term;
        result :=result+(2.0*power)/float(2*index+1);
        if old=result then
          exit;
        end if;
        old :=result;
      end loop;
      return result;
    elsif x=0.0 then
      return 1.0;
    else
      raise bounds_error;
    end if;
```

```
  end LN;
  ------------------------------------------------------------
  ------------------------------------------------------------

  function LN (x: in integer) return float is
  begin
    return(LN(float(x)));
  end LN;
  ------------------------------------------------------------
  ------------------------------------------------------------

  function "**" (a: in float; x: in float) return float is
    factorial : float := 1.0;
    result : float := 1.0;
    power : float := 1.0;
    old : float := 0.0;
  begin
    for limit in 1 .. integer'last loop
      power := power*(x*LN(a));
      factorial := factorial*float(limit);
      result := result+(power)/factorial;
      if old=result
      then
        exit;
      end if;
      old :=result;
    end loop;
    return result;
  end "**";
------------------------------------------------------------
------------------------------------------------------------

  function SQRT (a: in float) return float is
  begin
    if a=0.0 then
      return 0.0;
    else
      return a**0.5;
    end if;
  end SQRT;

end log;
```

APPENDIX Q

SOURCE LISTING
TRUCK SIMULATION MAIN PROGRAM
ADA REDESIGN
VADS COMPILER RELEASE V04.06

```
with SIMULATION_ROUTINES ; use SIMULATION_ROUTINES;

procedure TRUCK_SIMULATION is
   MEAN_INTER_ARRIVAL_TIME : FLOAT        := 1.0/3.0;
   MEAN_SERVICE_TIME       : FLOAT        := 0.25  ;
   STATS                   : STATISTICS;
   SERVICE_QUEUE           : QUEUE;
   RANDOM_NUMBER           : RANDOM_NUMBER_RECORD;
begin
   while STATS.REPETITION < 10 loop
      INITIALIZE (STATS,
                  SERVICE_QUEUE,
                  MEAN_INTER_ARRIVAL_TIME,
                  RANDOM_NUMBER);
      while STATS.TOTAL_DEPARTURES < 1500 loop
         if STATS.NEXT_ARRIVAL < STATS.NEXT_DEPARTURE then
            GENERATE_ARRIVAL (STATS,
                              SERVICE_QUEUE,
                              MEAN_INTER_ARRIVAL_TIME,
                              MEAN_SERVICE_TIME,
                              RANDOM_NUMBER);
         else
            GENERATE_DEPARTURE (STATS,
                                SERVICE_QUEUE,
                                MEAN_SERVICE_TIME,
                                RANDOM_NUMBER);
         end if;
      end loop;
      GENERATE_REPORT (STATS,
                       SERVICE_QUEUE,
                       MEAN_INTER_ARRIVAL_TIME,
                       MEAN_SERVICE_TIME,
                       RANDOM_NUMBER);
   end loop;

   exception
      when CONSTRAINT_ERROR =>
           GENERATE_REPORT (STATS,
                            SERVICE_QUEUE,
                            MEAN_INTER_ARRIVAL_TIME,
                            MEAN_SERVICE_TIME,
```

```
                                RANDOM_NUMBER);
        when others             =>
             GENERATE_REPORT (STATS,
                              SERVICE_QUEUE,
                              MEAN_INTER_ARRIVAL_TIME,
                              MEAN_SERVICE_TIME,
                              RANDOM_NUMBER);

end TRUCK_SIMULATION;
```

APPENDIX R

SOURCE LISTING
TRUCK SIMULATION ROUTINES PACKAGE
ADA REDESIGN
VADS COMPILER RELEASE V04.06

```
with TEXT_IO;
with LOG;      use LOG;

package SIMULATION_ROUTINES is

   package FLOAT_IO   is new TEXT_IO.FLOAT_IO   (FLOAT);
   package INTEGER_IO is new TEXT_IO.INTEGER_IO (INTEGER);

   type SEED_ARRAY is array (INTEGER range 1..10 ) of FLOAT;

   type RANDOM_NUMBER_RECORD is record
        NUMBER  :   FLOAT;
        SEEDS   :   SEED_ARRAY := ( 1 =>  567.0,
                                    2 =>  459.0,
                                    3 =>  561.0,
                                    4 =>  663.0,
                                    5 =>  613.0,
                                    6 =>  867.0,
                                    7 =>  969.0,
                                    8 => 1071.0,
                                    9 => 1173.0,
                                   10 => 2717.0);
        DSEED   :   FLOAT;
        COUNT   :   INTEGER;
   end record;

   type SIMPLE_ARRAY is
                      array (INTEGER range 1..100) of FLOAT;

   type STATISTICS is record
        CLOCK                   : FLOAT;
        NEXT_ARRIVAL            : FLOAT;
        NEXT_DEPARTURE          : FLOAT;
        TIME_LAST_EVENT         : FLOAT;
        SERVER_BUSY_TIME        : FLOAT;
        TOTAL_TIME_IN_SYSTEM    : FLOAT;
        TOTAL_ARRIVALS          : INTEGER;
        TOTAL_DEPARTURES        : INTEGER;
        MAX_Q_LENGTH            : INTEGER;
        FOUR_HOURS_IN_SYSTEM    : INTEGER;
```

```
        REPETITION              : INTEGER := 0;
    end record;

    type QUEUE is record
        ELEMENT : SIMPLE_ARRAY;
        LENGTH  : INTEGER := 0;
        IS_IDLE : BOOLEAN := TRUE;
    end record;

    procedure GENERATE_ARRIVAL
              (STATS                    : in out STATISTICS;
               SERVICE_QUEUE            : in out QUEUE;
               MEAN_INTER_ARRIVAL_TIME  : in     FLOAT;
               MEAN_SERVICE_TIME        : in     FLOAT;
               RANDOM_NUMBER            : in out
                                          RANDOM_NUMBER_RECORD);

    procedure GENERATE_DEPARTURE
              (STATS                : in out STATISTICS;
               SERVICE_QUEUE        : in out QUEUE;
               MEAN_SERVICE_TIME    : in     FLOAT;
               RANDOM_NUMBER        : in out
                                      RANDOM_NUMBER_RECORD);

    procedure INITIALIZE
              (STATS                    : in out STATISTICS;
               SERVICE_QUEUE            : in out QUEUE;
               MEAN_INTER_ARRIVAL_TIME  : in     FLOAT;
               RANDOM_NUMBER            : in out
                                          RANDOM_NUMBER_RECORD);

    procedure GENERATE_REPORT
              (STATS                    : in out STATISTICS;
               SERVICE_QUEUE            : in out QUEUE;
               MEAN_INTER_ARRIVAL_TIME  : in     FLOAT;
               MEAN_SERVICE_TIME        : in     FLOAT;
               RANDOM_NUMBER            : in out
                                          RANDOM_NUMBER_RECORD);

    procedure RAN
              (RANDOM_NUMBER :  in out RANDOM_NUMBER_RECORD);

end SIMULATION_ROUTINES;
-------------------------------------------------------------
-------------------------------------------------------------
package body SIMULATION_ROUTINES is

    procedure GENERATE_ARRIVAL
              (STATS                    : in out STATISTICS;
               SERVICE_QUEUE            : in out QUEUE;
               MEAN_INTER_ARRIVAL_TIME  : in     FLOAT;
               MEAN_SERVICE_TIME        : in     FLOAT;
               RANDOM_NUMBER            : in out
```

```
                                        RANDOM_NUMBER_RECORD) is

   SERVICE_TIME        : FLOAT;
   INTER_ARRIVAL_TIME  : FLOAT;

begin
   STATS.CLOCK           := STATS.NEXT_ARRIVAL;
   SERVICE_QUEUE.LENGTH  := SERVICE_QUEUE.LENGTH + 1;
   SERVICE_QUEUE.ELEMENT (SERVICE_QUEUE.LENGTH)
                         := STATS.CLOCK;
   if SERVICE_QUEUE.IS_IDLE then
      SERVICE_QUEUE.IS_IDLE := FALSE;
      RAN (RANDOM_NUMBER);
      SERVICE_TIME           :=  MEAN_SERVICE_TIME
                               * (-LN
                                  (RANDOM_NUMBER.NUMBER));
      STATS.NEXT_DEPARTURE   :=  STATS.CLOCK
                               + SERVICE_TIME;
   else
      STATS.SERVER_BUSY_TIME :=   STATS.SERVER_BUSY_TIME
                                + (STATS.CLOCK
                                -  STATS.TIME_LAST_EVENT);
   end if;
   STATS.TIME_LAST_EVENT := STATS.CLOCK;
   STATS.TOTAL_ARRIVALS  := STATS.TOTAL_ARRIVALS + 1;
   if SERVICE_QUEUE.LENGTH > STATS.MAX_Q_LENGTH tnen
      STATS.MAX_Q_LENGTH := SERVICE_QUEUE.LENGTH;
   end if;
   RAN (RANDOM_NUMBER);
   INTER_ARRIVAL_TIME :=    MEAN_INTER_ARRIVAL_TIME
                          * (-LN (RANDOM_NUMBER.NUMBER));
   STATS.NEXT_ARRIVAL :=    STATS.CLOCK
                          + INTER_ARRIVAL_TIME;
end GENERATE_ARRIVAL;
-------------------------------------------------------------
-------------------------------------------------------------
procedure GENERATE_DEPARTURE
                  (STATS               : in out STATISTICS;
                   SERVICE_QUEUE       : in out QUEUE;
                   MEAN_SERVICE_TIME   : in     FLOAT;
                   RANDOM_NUMBER       : in out
                                        RANDOM_NUMBER_RECORD) is

   TIME_IN_SYSTEM_THIS_DEPARTURE : FLOAT;
   SERVICE_TIME                  : FLOAT;

begin
   STATS.CLOCK                :=  STATS.NEXT_DEPARTURE;
   STATS.SERVER_BUSY_TIME     :=  STATS.SERVER_BUSY_TIME
                                + (STATS.CLOCK
                                - STATS.TIME_LAST_EVENT);
   STATS.TIME_LAST_EVENT          :=   STATS.CLOCK;
```

```
      TIME_IN_SYSTEM_THIS_DEPARTURE :=   STATS.CLOCK
                                 - SERVICE_QUEUE.ELEMENT(1);
      STATS.TOTAL_TIME_IN_SYSTEM
                           :=   STATS.TOTAL_TIME_IN_SYSTEM
                              + TIME_IN_SYSTEM_THIS_DEPARTURE;
      STATS.TOTAL_DEPARTURES     :=   STATS.TOTAL_DEPARTURES
                                    + 1;

      if TIME_IN_SYSTEM_THIS_DEPARTURE > 4.0 then
         STATS.FOUR_HOURS_IN_SYSTEM
                           := STATS.FOUR_HOURS_IN_SYSTEM + 1;
      end if;

      if SERVICE_QUEUE.LENGTH - 1 = 0 then    -- if queue
                                              -- will be
                                              -- empty
                                              -- after this
                                              -- departure
         SERVICE_QUEUE.LENGTH  := 0;
         SERVICE_QUEUE.IS_IDLE := TRUE;
         STATS.NEXT_DEPARTURE  := 1.0e30;
      else
         for INDEX in 1..SERVICE_QUEUE.LENGTH - 1 loop
            SERVICE_QUEUE.ELEMENT (INDEX)
                        := SERVICE_QUEUE.ELEMENT(INDEX + 1);
         end loop;
         SERVICE_QUEUE.LENGTH  := SERVICE_QUEUE.LENGTH - 1;
         RAN (RANDOM_NUMBER);
         SERVICE_TIME          :=  MEAN_SERVICE_TIME
                                 * (-LN
                                   (RANDOM_NUMBER.NUMBER));
         STATS.NEXT_DEPARTURE  :=  STATS.CLOCK
                                 + SERVICE_TIME;
      end if;
   end GENERATE_DEPARTURE;
   ------------------------------------------------------------
   ------------------------------------------------------------
   procedure INITIALIZE
                 (STATS                       : in out
                                                STATISTICS;
                  SERVICE_QUEUE               : in out QUEUE;
                  MEAN_INTER_ARRIVAL_TIME     : in     FLOAT;
                  RANDOM_NUMBER               : in out
                                          RANDOM_NUMBER_RECORD) is
      ARRIVAL_TIME  : FLOAT;
   begin
      STATS.REPETITION     := STATS.REPETITION + 1;
      RANDOM_NUMBER.DSEED  := RANDOM_NUMBER.SEEDS
                              (STATS.REPETITION);
      RANDOM_NUMBER.COUNT  := 0;
      RAN (RANDOM_NUMBER);
      ARRIVAL_TIME         :=  MEAN_INTER_ARRIVAL_TIME
```

```
                              * (-LN (RANDOM_NUMBER.NUMBER));
   STATS.CLOCK                    := 0.0;
   STATS.TIME_LAST_EVENT          := 0.0;
   STATS.SERVER_BUSY_TIME         := 0.0;
   STATS.TOTAL_TIME_IN_SYSTEM     := 0.0;
   STATS.TOTAL_ARRIVALS           := 0   ;
   STATS.TOTAL_DEPARTURES         := 0   ;
   STATS.MAX_Q_LENGTH             := 0   ;
   STATS.FOUR_HOURS_IN_SYSTEM     := 0   ;
   STATS.NEXT_ARRIVAL             :=  STATS.CLOCK
                                    + ARRIVAL_TIME;
   STATS.NEXT_DEPARTURE           := 1.0e30;
   SERVICE_QUEUE.LENGTH           := 0;
   SERVICE_QUEUE.IS_IDLE          := TRUE;
end INITIALIZE;
-------------------------------------------------------------
-------------------------------------------------------------
procedure GENERATE_REPORT
                (STATS                        : in out
                                                STATISTICS;
                 SERVICE_QUEUE                : in out QUEUE;
                 MEAN_INTER_ARRIVAL_TIME      : in     FLOAT;
                 MEAN_SERVICE_TIME            : in     FLOAT;
                 RANDOM_NUMBER                : in out
                                    RANDOM_NUMBER_RECORD) is
   TEMP : FLOAT;
begin
   NEW_LINE;
   NEW_LINE;
   PUT ("RANDOM NUMBER GENERATOR SEED");
   PUT (RANDOM_NUMBER.SEEDS (STATS.REPETITION));
   NEW_LINE;
   PUT ("MEAN INTERARRIVAL TIME = ");
   PUT (MEAN_INTER_ARRIVAL_TIME);
   NEW_LINE;
   PUT ("MEAN SERVICE TIME = ");
   PUT (MEAN_SERVICE_TIME);
   NEW_LINE;
   NEW_LINE;
   PUT ("PROPORTION OF TIME DOCK CREW IS BUSY = ");
   TEMP := STATS.SERVER_BUSY_TIME / STATS.CLOCK;
   PUT (TEMP);
   NEW_LINE;
   NEW_LINE;
   PUT ("MAXIMUM LENGTH OF WAITING LINE = ");
   PUT (STATS.MAX_Q_LENGTH);
   NEW_LINE;
   NEW_LINE;
   PUT ("AVERAGE TIME TO TRANSIT SYSTEM = ");
   TEMP :=          STATS.TOTAL_TIME_IN_SYSTEM
          / FLOAT(STATS.TOTAL_DEPARTURES);
   PUT (TEMP);
```

```
      NEW_LINE;
      NEW_LINE;
      PUT ("PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS
           =");
      TEMP :=     FLOAT (STATS.FOUR_HOURS_IN_SYSTEM)
              /   FLOAT (STATS.TOTAL_DEPARTURES);
      PUT (TEMP);
      NEW_LINE;
      NEW_LINE;
      PUT ("SIMULATION RUN LENGTH = ");
      PUT (STATS.CLOCK);
      PUT ("  HOURS");
      NEW_LINE;
      NEW_LINE;
      PUT ("NUMBER OF TRUCKS UNLOADED = ");
      PUT (STATS.TOTAL_DEPARTURES);
      NEW_LINE;
      NEW_LINE;
      PUT ("NUMBER OF RANDOM NUMBERS USED = ");
      PUT (RANDOM_NUMBER.COUNT);
      NEW_LINE;
      NEW_LINE;
      PUT ("AVERAGE NUMBER OF UNITS IN SYS. = ");
      TEMP := STATS.TOTAL_TIME_IN_SYSTEM / STATS.CLOCK;
      PUT (TEMP);
      NEW_LINE;
      NEW_LINE;
      PUT ("TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
           ");
      PUT (STATS.TOTAL_TIME_IN_SYSTEM);
      NEW_LINE;
      NEW_LINE;
      PUT ("AVERAGE NUMBER OF ARRIVALS PER HR = ");
      TEMP := FLOAT (STATS.TOTAL_ARRIVALS) / STATS.CLOCK;
      PUT (TEMP);
      NEW_LINE;
      NEW_LINE;
   end GENERATE_REPORT;

-------------------------------------------------------------
-------------------------------------------------------------
  procedure RAN (RANDOM_NUMBER :  in out
                                    RANDOM_NUMBER_RECORD) is
     type DIGITS_6 is digits 6;

     tmpint  : integer;
     tmpreal : DIGITS_6;
     TEMP    : DIGITS_6;
     SEED    : DIGITS_6;

  begin
     SEED    := DIGITS_6 (RANDOM_NUMBER.DSEED);
```

```
        tmpreal := SEED * 3.141592;
        tmpint  := integer(tmpreal);
        TEMP    := tmpreal - DIGITS_6 (tmpint);

        if TEMP >= 0.5 then
           tmpint  := tmpint + 1;
           tmpreal := tmpreal - DIGITS_6 (tmpint);
        else
           tmpreal := TEMP;
        end if;

        if tmpreal < 0.0 then
          tmpreal := -tmpreal;
        end if;
        tmpreal := 2.0 * tmpreal;
        RANDOM_NUMBER.NUMBER := FLOAT (tmpreal);
        RANDOM_NUMBER.DSEED  := FLOAT (tmpreal);
        RANDOM_NUMBER.COUNT  := RANDOM_NUMBER.COUNT + 1;
    end RAN;
end SIMULATION_ROUTINES;
```

APPENDIX S

SOURCE LISTING
LIBRARY MAINTENANCE PROGRAM
ORIGINAL PASCAL VERSION

```
program liblist(input, output, libfile);

type chararr= array[1..20] of char;
libptr = ^liblist;
  liblist =
          record
                NEXT:  libptr;
                NAME:  chararr;
              AUTHOR:  chararr;
              CALLNO:  integer;
        end; (*RECORD*)

var   FRONT,
       BOOK:   libptr;
   INCALLNO,
       INDX:   integer;
  SELECTION:   char;
    libfile:   text;

procedure insert(BOOK: libptr);

var P,Q: libptr;

begin (*insert*)
        if FRONT = nil then
          FRONT := BOOK
       else
            if FRONT^.CALLNO>BOOK^.CALLNO then
                 begin(*INSERT AT FRONT*)
                      BOOK^.NEXT := FRONT;
                      FRONT := BOOK
                 end
            else
                 begin  (*INSERT IN MIDDLE*)
                      P := FRONT;
                      Q := FRONT;
                      while(P^.NEXT<>nil) and (P = Q) do
                          begin(*TRAVERSE*)
                               P := P^.NEXT;
                               if P^.CALLNO>BOOK^.CALLNO then
                                     begin (*ATTACH*)
                                          Q^.NEXT := BOOK;
```

```
                                        BOOK^.NEXT := P
                                    end
                                else
                                    Q := P
                            end; (*TRAVERSE*)
                        if(P^.NEXT = nil) and
                                (P^.CALLNO<BOOK^.CALLNO) then
                            (*ATTACH AT END*)
                            P^.NEXT := BOOK
                    end (*INSERT IN MIDDLE*)
        end; (*INSERT*)

    procedure delete(CALLNO: integer);

        var   P,Q: libptr;
           DELETED: boolean;

        begin (*DELETE*)
            DELETED := FALSE;
            if FRONT = nil then
                writeln('NOTHING TO DELETE.')
            else
                if FRONT^.CALLNO = CALLNO then
                    begin (*DELETE FIRST ELEMENT*)
                        FRONT := FRONT^.NEXT;
                        DELETED := TRUE
                    end (*DELETE FIRST ELEMENT*)
                else
                    begin (*SEARCH LIST*)
                        P := FRONT;
                        Q := FRONT;
                        while (P^.NEXT<>nil) and (P = Q) and
                              (P^.CALLNO<CALLNO) and
                                          (DELETED=FALSE)do
                            begin (*TRAVERSE and DELETE*)
                                P := P^.NEXT;
                                if P^.CALLNO = CALLNO then
                                    begin (*DELETE BOOK*)
                                        Q^.NEXT := P^.NEXT;
                                        DELETED := TRUE
                                    end (*DELETE BOOK*)
                                else
                                    Q := P
                            end; (*TRAVERSE and DELETE*)
                         if DELETED = FALSE then
                               writeln('NO SUCH BOOK');
                               writeln
                    end (*SEARCH LIST*)
        end; (*DELETE*)

    procedure readfile;
```

```
    var INDX: integer;
        BOOK: libptr;

    begin (*readfile*)
        reset(libfile);

        while not eof(libfile) do
            begin (*READ BOOK*)
                new(BOOK);
                for INDX := 1 to 20 do
                    read(libfile, BOOK^.NAME[INDX]);
                readln(libfile);
                for INDX := 1 to 20 do
                    read(libfile, BOOK^.AUTHOR[INDX]);
                readln(libfile);
                readln(libfile, BOOK^.CALLNO);
                insert(BOOK)
            end (*READ BOOK*)
    end; (*readFILE*)

procedure writefile;

    var   P:  libptr;
       INDX:  integer;

    begin (*writefile*)
        rewrite(libfile);
        P := FRONT;
        while P<>nil do
            begin (*write BOOK*)
                for INDX := 1 to 20 do
                    write(libfile, P^.NAME[INDX]);
                writeln(libfile);
                for INDX := 1 to 20 do
                    write(libfile, P^.AUTHOR[INDX]);
                    writeln(libfile);
                    writeln(libfile, P^.CALLNO);
                    P := P^.NEXT
            end (*WRITE BOOK*)
    end; (*writefile*)

procedure viewfile;

    var  INDX: integer;
         NAME, AUTHOR: chararr;
         CALLNO: integer;

    begin (*viewfile*)
        reset(libfile);
        while not eof(libfile) do
            begin (*view libfile*)
                for INDX := 1 to 20 do
```

```
                    begin (*loop*)
                        read(libfile,NAME[INDX]);
                        write(NAME[INDX]);
                    end;
                    readln(libfile);
                    writeln;
                for INDX := 1 to 20 do
                    begin (*loop*)
                        read(libfile,AUTHOR[INDX]);
                        write(AUTHOR[INDX]);
                    end;
                    readln(libfile);
                    writeln;
                readln(libfile,CALLNO);
                writeln(CALLNO);
            end; (*view libfile*)
            writeln;
            writeln('END OF LIBRARY FILE');
    end; (*viewfile*)


begin (*liblist*)
    FRONT := nil;
    readfile;
    writeln ('WOULD YOU LIKE TO INSERT OR DELETE A BOOK OR
              VIEW THE FILE?');
    write ('TYPE I OR D OR V: ');
    readln(SELECTION);
    writeln;
    while SELECTION<>'F' do
        begin (*UPDATE LIST*)
            if SELECTION = 'I' then
                begin (*READ, INSERT BOOK*)
                new(BOOK);
                writeln('TYPE THE NAME OF THE BOOK: ');
                for INDX := 1 to 20 do
                    if not eoln then
                        read(BOOK^.NAME[INDX])
                    else
                        BOOK^.NAME[INDX] := ' ';
                readln;
                writeln;
                writeln('TYPE THE NAME OF THE AUTHOR:');
                for INDX := 1 to 20 do
                    if not eoln then
                        read(BOOK^.AUTHOR[INDX])
                    else
                        BOOK^.AUTHOR[INDX] := ' ';
                readln;
                writeln;
                writeln ('TYPE THE CALL NUMBER OF THE
```

```
                                BOOK:');
                        readln (BOOK^.CALLNO);
                        writeln;
                        insert(BOOK);
                    end; (*READ, INSERT BOOK*)

            if SELECTION = 'D' then
                begin (*GET NUMBER, DELETE BOOK*)
                    write('TYPE THE CALL NUMBER OF THE BOOK:');
                    readln(INCALLNO);
                    writeln;
                    delete(INCALLNO);
                end; (*GET NUMBER, DELETE BOOK*)

            if SELECTION = 'V' then
                begin (*TO VIEW FILE*)
                    writefile;
                    viewfile;
                    writeln;
                end; (*TO VIEW FILE*)

            write('TYPE I TO INSERT, D TO DELETE, OR V TO VIEW
                   FILE, OR F TO FINISH:');
            readln(SELECTION);
            writeln
        end; (*UPDATE LIST*)
    writefile;
    writeln('LIBRARY FILE IS NOW UPDATED');
    writeln; writeln
    end. (*liblist*)
```

APPENDIX T

SOURCE LISTING
LIBRARY MAINTENANCE PROGRAM
ADA LINE-BY-LINE TRANSLATION
VADS COMPILER RELEASE V04.06

```
------------------------------------------------------------
------------------------------------------------------------
-- LINE BY LINE TRANSLATION OF THE LIBLIST PASCAL PROGRAM
-- INTO ADA.
------------------------------------------------------------
------------------------------------------------------------
with text_io; use text_io;

procedure lib is

  package int_io is new integer_io(integer);
  use int_io;

  type chararr is array (integer range 1 .. 20) of
                                                  character;
  type liblist;
  type libptr is access liblist;

  type liblist is
    record
         NEXT : libptr;
         NAME : chararr;
       AUTHOR : chararr;
       CALLNO : integer;
    end record;

  FRONT, BOOK : libptr;
  INCALLNO    : integer;
  SELECTION : character;

  libfile : text_io.file_type;

------------------------------------------------------------

procedure insert (BOOK : libptr) is

  P,Q : libptr;

begin
```

```
  if FRONT = null then
    FRONT := BOOK;

  else

    if FRONT.CALLNO > BOOK.CALLNO then
      BOOK.NEXT := FRONT;
      FRONT := BOOK;

    else

      P := FRONT;
      Q := FRONT;
      while (P.NEXT /= null) and (P = Q) loop
        P := P.NEXT;
        if P.CALLNO > BOOK.CALLNO then
          Q.NEXT := BOOK;
          BOOK.NEXT := P;

        else

          Q := P;
        end if;
      end loop;
      if ((P.NEXT = null) and (P.CALLNO < BOOK.CALLNO)) then
        P.NEXT := BOOK;
      end if;
    end if;
  end if;
end insert;

------------------------------------------------------------

procedure delete (CALLNO : in INTEGER) is

  P, Q : libptr;
  DELETED : BOOLEAN := FALSE;

begin

  if FRONT = null then
    put("NOTHING TO DELETE");
    new_line;

  else

    if FRONT.CALLNO = CALLNO then
      FRONT := FRONT.NEXT;
      DELETED := TRUE;

    else
```

```
      P := FRONT;
      Q := FRONT;
      while (P.NEXT /= null) and (P = Q) and
                (P.CALLNO<CALLNO) and (DELETED = FALSE) loop
        P := P.NEXT;
        if P.CALLNO = CALLNO then
          Q.NEXT := P.NEXT;
          DELETED := TRUE;

        else

          Q := P;
        end if;
      end loop;

      if DELETED = FALSE then
        put("NO SUCH BOOK");
        new_line;
      end if;

    end if;
  end if;

end delete;

-------------------------------------------------------------

procedure readfile is

  BOOK : libptr;

begin

  reset(libfile);

  while not END_OF_FILE(libfile) loop
    BOOK := new liblist;
    for I IN 1 .. 20 loop
      get (libfile, BOOK.NAME(I));
    end loop;
    skip_line(libfile);

    for I in 1 .. 20 loop
      get (libfile, BOOK.AUTHOR(I));
    end loop;
    skip_line(libfile);

    get (libfile, BOOK.CALLNO);
    skip_line(libfile);

  insert(BOOK);
  end loop;
```

```
end readfile;

-------------------------------------------------------------
procedure writefile is

  P : libptr;

begin

  create(libfile, out_file,"libfile");
  P := FRONT;
  while (P /= null) loop
    for I in 1 .. 20 loop
      put(libfile, P.NAME(I));
    end loop;

    new_line(libfile);

    for I in 1 .. 20 loop
      put(libfile, P.AUTHOR(I));
    end loop;

    new_line(libfile);

    put(libfile, P.CALLNO); new_line(libfile);

    P := P.NEXT;
  end loop;
  close(libfile);
end writefile;

-------------------------------------------------------------

procedure view_libfile is

  NAME : chararr;
  AUTHOR : chararr;
  CALLNO : integer;

begin

  reset(libfile);

  while not END_OF_FILE(libfile) loop

    for I in 1 .. 20 loop
      get(libfile,NAME(I));
      put(NAME(I));
    end loop;
    skip_line(libfile);
```

```
    new_line;

    for I in 1 .. 20 loop
      get(libfile,AUTHOR(I));
      put(AUTHOR(I));
    end loop;
    skip_line(libfile);
    new_line;

    get(libfile,CALLNO);
    put(CALLNO);
    skip_line(libfile);
    new_line;

  end loop;

  new_line;
  put("END OF LIBRARY FILE");
  new_line;
  new_line;

end view_libfile;

------------------------------------------------------------
------------------------------------------------------------
-- MAIN PROGRAM BODY

begin

  open(libfile, in_file,"libfile");
  FRONT := null;
  readfile;

  put("WOULD YOU LIKE TO INSERT OR DELETE A BOOK OR VIEW THE
FILE?"); new_line;
  put("TYPE I OR D OR V: ");
  get(SELECTION);
  skip_line;

  while SELECTION /= 'F' loop

  if SELECTION = 'I' then
    BOOK := new liblist;
    put("TYPE THE NAME OF THE BOOK: ");
    new_line;

  for I in 1 .. 20 loop
    if not END_OF_LINE then
      get(BOOK.NAME(I));

    else
```

```
        BOOK.NAME(I) := ' ';
      end if;
    end loop;
    new_line;
    skip_line;

    put("TYPE THE NAME OF THE AUTHOR: ");
    new_line;
    for I in 1 .. 20 loop
      if not END_OF_LINE then
        get(BOOK.AUTHOR(I));

      else

        BOOK.AUTHOR(I) := ' ';
      end if;
    end loop;
    new_line;

    put("TYPE THE CALLNO OF THE BOOK: ");
    new_line;
    get(BOOK.CALLNO);
    new_line;

    insert(BOOK);

  elsif SELECTION = 'D' then
    put("TYPE THE CALL NUMBER OF THE BOOK:");
    get(INCALLNO);
    new_line;

    delete(INCALLNO);
    elsif SELECTION = 'V' then
    close(libfile);
    writefile;
    open(libfile, in_file, "libfile");
    view_libfile;
    new_line;
  end if;
  put("TYPE I TO INSERT, D TO DELETE, V TO VIEW FILE OR F TO
       FINISH: ");
  get(SELECTION);
  new_line;
  skip_line;
  end loop;
  close(libfile);
  writefile;
  put("LIBRARY FILE IS NOW UPDATED");
  new_line; new_line;
  end lib;
```

APPENDIX U

SOURCE LISTING
LIBRARY MAINTENANCE MAIN PROGRAM
ADA REDESIGN
VADS COMPILER RELEASE V04.06

```
with TEXT_IO;   use TEXT_IO;
with LIB_LIST;  use LIB_LIST;
procedure LIB_MAIN is
   ACTION   : CHARACTER;
   LIB_FILE : LIB_IO.FILE_TYPE;
begin
   LIB_IO.OPEN (FILE => LIB_FILE,
                MODE => LIB_IO.INOUT_FILE,
                NAME => "lib_file.dat");
   NEW_LINE;
   PUT ("LIBRARY MAINTENANCE PROGRAM");
   NEW_LINE;
   NEW_LINE;
   loop
      NEW_LINE;
      PUT ("What do you want to do?");      NEW_LINE;
      PUT ("    'I' = insert a book");      NEW_LINE;
      PUT ("    'D' = delete a book");      NEW_LINE;
      PUT ("    'P' = print library list"); NEW_LINE;
      PUT ("    'Q' = quit");               NEW_LINE;
      GET (ACTION);
      exit when ACTION = 'Q';
      case ACTION is
         when 'I'      => INSERT_BOOK (LIB_FILE);
         when 'D'      => DELETE_BOOK (LIB_FILE);
         when 'P'      => PRINT_BOOK_LIST (LIB_FILE);
         when others   => PUT_LINE ("INVALID RESPONSE");
      end case;
   end loop;
   LIB_IO.CLOSE (FILE => LIB_FILE);
exception
   when NUMERIC_ERROR =>
      LIB_IO.CLOSE (FILE => LIB_FILE);
      PUT ("FILE CLOSED");
      raise;
   when others        =>
      LIB_IO.CLOSE (FILE => LIB_FILE);
      PUT ("FILE CLOSED");
      raise;
end LIB_MAIN;
```

APPENDIX V

SOURCE LISTING
LIBRARY MAINTENANCE ROUTINES PACKAGE
ADA REDESIGN
VADS COMPILER RELEASE V04.06

```
with DIRECT_IO;
with TEXT_IO;   use TEXT_IO;
package LIB_LIST is

   type LIBRARY_BOOK is record
      NEXT_BOOK   : POSITIVE;
      TITLE       : STRING (1..20);
      AUTHOR      : STRING (1..20);
      CALL_NUMBER : POSITIVE;
   end record;

   package LIB_IO is new DIRECT_IO  (ELEMENT_TYPE =>
                                          LIBRARY_BOOK);
   package INT_IO is new INTEGER_IO (POSITIVE);
   use LIB_IO;
   use INT_IO;

   procedure INSERT_BOOK (LIB_FILE : in LIB_IO.FILE_TYPE);
   procedure DELETE_BOOK (LIB_FILE : in LIB_IO.FILE_TYPE);
   procedure PRINT_BOOK_LIST (LIB_FILE : in
                                    LIB_IO.FILE_TYPE);

end LIB_LIST;
-------------------------------------------------------------
-------------------------------------------------------------
package body LIB_LIST is

   procedure INSERT_BOOK (LIB_FILE : in LIB_IO.FILE_TYPE) is

      NEW_BOOK        : LIBRARY_BOOK;
      THIS_BOOK       : LIBRARY_BOOK;
      PREV_BOOK       : LIBRARY_BOOK;
      PREV_BOOK_INDEX : LIB_IO.POSITIVE_COUNT;

   begin
      NEW_LINE;
      PUT ("Enter book title");
      NEW_LINE;
      SKIP_LINE;
      for I in NEW_BOOK.TITLE'range loop
```

```
   if not END_OF_LINE then
      GET (NEW_BOOK.TITLE(I));
   else
      NEW_BOOK.TITLE(I) := ' ';
   end if;
end loop;
NEW_LINE;
PUT ("Enter author name");
NEW_LINE;
SKIP_LINE;
for I in NEW_BOOK.AUTHOR'range loop
   if not END_OF_LINE then
      GET (NEW_BOOK.AUTHOR(I));
   else
      NEW_BOOK.AUTHOR(I) := ' ';
   end if;
end loop;
NEW_LINE;
PUT ("Enter call number");
NEW_LINE;
GET (NEW_BOOK.CALL_NUMBER);
NEW_LINE;
READ (FILE => LIB_FILE,
      ITEM => THIS_BOOK,
      FROM => 1);
if THIS_BOOK.NEXT_BOOK = POSITIVE'last then
   NEW_BOOK.NEXT_BOOK  :=  POSITIVE'last;
   THIS_BOOK.NEXT_BOOK :=  POSITIVE(SIZE (LIB_FILE)
                           + 1);
   WRITE (FILE => LIB_FILE,
          ITEM => NEW_BOOK,
          TO   =>
          LIB_IO.POSITIVE_COUNT(THIS_BOOK.NEXT_BOOK));
   WRITE (FILE => LIB_FILE,
          ITEM => THIS_BOOK,
          TO   => 1);
else
   PREV_BOOK_INDEX := 1;
   PREV_BOOK       := THIS_BOOK;
   loop
      READ (FILE => LIB_FILE,
            ITEM => THIS_BOOK,
            FROM =>
                    LIB_IO.POSITIVE_COUNT
                   (THIS_BOOK.NEXT_BOOK));
      if THIS_BOOK.CALL_NUMBER > NEW_BOOK.CALL_NUMBER
                                                 then
         NEW_BOOK.NEXT_BOOK  := PREV_BOOK.NEXT_BOOK;
         PREV_BOOK.NEXT_BOOK :=POSITIVE(SIZE(LIB_FILE)
                               + 1);
         WRITE (FILE => LIB_FILE,
                ITEM => PREV_BOOK,
```

```
                    TO   => PREV_BOOK_INDEX);
                WRITE (FILE => LIB_FILE,
                       ITEM => NEW_BOOK,
                       TO   =>
                              LIB_IO.POSITIVE_COUNT
                             (PREV_BOOK.NEXT_BOOK));
                exit;
             else
                if THIS_BOOK.NEXT_BOOK = POSITIVE'last then
                   NEW_BOOK.NEXT_BOOK  := POSITIVE'last;
                   THIS_BOOK.NEXT_BOOK := POSITIVE(SIZE
                                          (LIB_FILE) + 1);
                   WRITE (FILE => LIB_FILE,
                          ITEM => THIS_BOOK,
                          TO   => INDEX (LIB_FILE) -1);
                   WRITE (FILE => LIB_FILE,
                          ITEM => NEW_BOOK,
                          TO   =>
                                  LIB_IO.POSITIVE_COUNT
                                 (THIS_BOOK.NEXT_BOOK));
                   exit;
                else
                   PREV_BOOK_INDEX := INDEX (LIB_FILE) - 1;
                   PREV_BOOK       := THIS_BOOK;
                end if;
             end if;
          end loop;
       end if;
    end INSERT_BOOK;
-------------------------------------------------------------
-------------------------------------------------------------

    procedure DELETE_BOOK (LIB_FILE : in LIB_IO.FILE_TYPE) is
       DELETED_BOOK      : LIBRARY_BOOK;
       THIS_BOOK         : LIBRARY_BOOK;
       PREV_BOOK         : LIBRARY_BOOK;
       PREV_BOOK_INDEX   : LIB_IO.POSITIVE_COUNT;
       BOOK_NOT_FOUND    : BOOLEAN := TRUE;
    begin
       NEW_LINE;
       PUT ("Enter call number of book to be deleted.");
       NEW_LINE;
       GET (DELETED_BOOK.CALL_NUMBER);
       READ (FILE => LIB_FILE,
             ITEM => THIS_BOOK,
             FROM => 1);
       PREV_BOOK_INDEX := 1;
       PREV_BOOK       := THIS_BOOK;
       while THIS_BOOK.NEXT_BOOK /= POSITIVE'last loop
          READ (FILE => LIB_FILE,
                ITEM => THIS_BOOK,
                FROM => LIB_IO.POSITIVE_COUNT
```

```
                        (THIS_BOOK.NEXT_BOOK));
            if THIS_BOOK.CALL_NUMBER = DELETED_BOOK.CALL_NUMBER
                                                          then
               if THIS_BOOK.NEXT_BOOK = POSITIVE'last then
                  PREV_BOOK.NEXT_BOOK := POSITIVE'last;
               else
                  PREV_BOOK.NEXT_BOOK := THIS_BOOK.NEXT_BOOK;
               end if;
               WRITE (FILE => LIB_FILE,
                      ITEM => PREV_BOOK,
                      TO   => PREV_BOOK_INDEX);
               BOOK_NOT_FOUND := FALSE;
               exit;
            end if;
            PREV_BOOK_INDEX := INDEX (LIB_FILE) - 1;
            PREV_BOOK       := THIS_BOOK;
         end loop;
         if BOOK_NOT_FOUND then
            NEW_LINE;
            PUT ("Book Not Found");
            NEW_LINE;
         end if;
      end DELETE_BOOK;
-------------------------------------------------------------
-------------------------------------------------------------

      procedure PRINT_BOOK_LIST (LIB_FILE : in
                                            LIB_IO.FILE_TYPE) is
         BOOK : LIBRARY_BOOK;
      begin
         READ (FILE => LIB_FILE,
               ITEM => BOOK,
               FROM => 1);
         while BOOK.NEXT_BOOK /= INTEGER'last loop
            READ (FILE => LIB_FILE,
                  ITEM => BOOK,
                  FROM => LIB_IO.POSITIVE_COUNT
                          (BOOK.NEXT_BOOK));
            NEW_LINE;
            PUT (BOOK.CALL_NUMBER);
            PUT ("  ");
            PUT (BOOK.TITLE);
            PUT (" by ");
            PUT (BOOK.AUTHOR);
         end loop;
         NEW_LINE;
      end PRINT_BOOK_LIST;
   end LIB_LIST;
```

APPENDIX W

SOURCE LISTING
LIBRARY MAINTENANCE DATA FILE CREATION PROGRAM
ADA REDESIGN
VADS COMPILER RELEASE V04.06

```
with LIB_LIST; use LIB_LIST;
procedure MAKE_FILE is
   BOOK     : LIBRARY_BOOK;
   LIB_FILE : LIB_IO.FILE_TYPE;
begin
   LIB_IO.CREATE (FILE => LIB_FILE,
                  MODE => LIB_IO.INOUT_FILE,
                  NAME => "lib_file.dat");
   BOOK.NEXT_BOOK    := 2;
   BOOK.AUTHOR       := "POINTS TO START OF  ";
   BOOK.TITLE        := "CHAIN               ";
   BOOK.CALL_NUMBER := POSITIVE'last;
   LIB_IO.WRITE (FILE => LIB_FILE,
                 ITEM => BOOK,
                 TO   => 1);
   BOOK.NEXT_BOOK    := POSITIVE'last;
   BOOK.AUTHOR       := "TOLSTOY, COUNT LEO  ";
   BOOK.TITLE        := "ANNA KARENINA       ";
   BOOK.CALL_NUMBER := 10;
   LIB_IO.WRITE (FILE => LIB_FILE,
                 ITEM => BOOK,
                 TO   => 2);
   LIB_IO.CLOSE (FILE => LIB_FILE);
end MAKE_FILE;
```

APPENDIX X

OUTPUT LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
FORTRAN 4 VERSION

```
Trapezoidal integration with end correction

      1  4.44444
      2  1.70535
      4  2.13427
      8  2.19111
     16  2.19675
     32  2.19719
     64  2.19722
    128  2.19723

Area =   2.19723
```

APPENDIX Y

OUTPUT LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
ADA LINE-BY-LINE TRANSLATION
TELESOFT-ADA COMPILER VERSION 1.5

```
Trapezoidal integration with end correction

    1  4.4444446E+00
    2  1.7053498E+00
    4  2.1342740E+00
    8  2.1911082E+00
   16  2.1967544E+00
   32  2.1971931E+00
   64  2.1972231E+00
  128  2.1972255E+00

Area = 2.1972255E+00
```

APPENDIX Z

OUTPUT LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
ADA LINE-BY-LINE TRANSLATION USING DEFAULT FLOAT PRECISION
VADS COMPILER RELEASE V04.06

```
Trapezoidal integration with end correction

          1   4.44444444E+00
          2   1.70534979E+00
          4   2.13427396E+00
          8   2.19110817E+00
         16   2.19675417E+00
         32   2.19719294E+00
         64   2.19722256E+00
        128   2.19722445E+00

Area = 2.19722445E+00
```

APPENDIX AA

OUTPUT LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
ADA LINE-BY-LINE TRANSLATION USING SIX DIGIT PRECISION
VADS COMPILER RELEASE V04.06

```
Trapezoidal integration with end correction

          1  4.44444E+00
          2  1.70535E+00
          4  2.13427E+00
          8  2.19111E+00
         16  2.19675E+00
         32  2.19719E+00
         64  2.19722E+00
        128  2.19723E+00

Area = 2.19723E+00
```

APPENDIX BB

OUTPUT LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
ADA REDESIGN USING DEFAULT FLOAT PRECISION
VADS COMPILER RELEASE V04.06

```
TRAPEZOIDAL INTEGRATION
          1 4.44444444E+00
          2 1.70534979E+00
          4 2.13427396E+00
          8 2.19110817E+00
         16 2.19675417E+00
         32 2.19719294E+00
         64 2.19722256E+00
        128 2.19722445E+00
AREA =  2.19722445E+00
```

APPENDIX CC

OUTPUT LISTING
TRAPEZOIDAL INTEGRATION PROGRAM
ADA REDESIGN USING SIX DIGIT PRECISION
VADS COMPILER RELEASE V04.06

```
TRAPEZOIDAL INTEGRATION
          1 4.44444E+00
          2 1.70535E+00
          4 2.13427E+00
          8 2.19111E+00
         16 2.19675E+00
         32 2.19719E+00
         64 2.19722E+00
        128 2.19723E+00
AREA =  2.19723E+00
```

APPENDIX DD

OUTPUT LISTING
TRUCK SIMULATION PROGRAM
FORTRAN 4 VERSION USING 3500 ELEMENT ARRAY

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.56700000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.73

MAXIMUM LENGTH OF WAITING LINE = 15

AVERAGE TIME TO TRANSIT SYS. 0.80 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.01

SIMULATION RUN LENGTH 509.36 HOURS..

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3007

AVERAGE NUMBER OF UNITS IN SYS.= 2.370

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1207.228
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.9547

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.45900000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.73

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 0.71 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 533.25 HOURS..

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3004

AVERAGE NUMBER OF UNITS IN SYS.= 1.985

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1058.522
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8167

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.56100000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.72

MAXIMUM LENGTH OF WAITING LINE = 13

AVERAGE TIME TO TRANSIT SYS. 0.79 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 529.22 HOURS..

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3008

AVERAGE NUMBER OF UNITS IN SYS.= 2.253

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1192.372
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.3457

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.66300000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.72

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 0.69 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 534.84 HOURS..

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3004

AVERAGE NUMBER OF UNITS IN SYS.= 1.937

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1035.763
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8083

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.61300000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.74

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 0.71 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 530.41 HOURS..

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3002

AVERAGE NUMBER OF UNITS IN SYS.= 1.996

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1058.438
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8299

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.86700000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.72

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 0.70 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 525.60 HOURS..

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3004

AVERAGE NUMBER OF UNITS IN SYS.= 1.996

```
 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=    1049.182
 (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=         2.8577


TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER
QUEUE.


 DSEED =           0.96900000d+03
 MEAN ARRIVAL TIME(MIAT) =          0.3333
 MEAN SERVICE TIME(MSVT) =          0.2500


 PROPORTION OF TIME DOCK CREW IS BUSY =     0.72

 MAXIMUM LENGTH OF WAITING LINE =       12

 AVERAGE TIME TO TRANSIT SYS.    0.80   HOURS..

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
 SYSTEM  0.00

 SIMULATION RUN LENGTH  525.89   HOURS..

 NUMBER OF TRUCKS UNLOADED =     1500

 NUMBER OF RANDOM NUMBERS USED =       3002

 AVERAGE NUMBER OF UNITS IN SYS.=       2.280

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=    1198.781
 (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=         2.8542


TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER
QUEUE.


 DSEED =           0.10710000d+04
 MEAN ARRIVAL TIME(MIAT) =          0.3333
 MEAN SERVICE TIME(MSVT) =          0.2500
```

PROPORTION OF TIME DOCK CREW IS BUSY = 0.71

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 0.69 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 530.05 HOURS..

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3002

AVERAGE NUMBER OF UNITS IN SYS.= 1.957

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1037.296
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8318

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.11730000d+04
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.72

MAXIMUM LENGTH OF WAITING LINE = 12

AVERAGE TIME TO TRANSIT SYS. 0.81 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 531.09 HOURS..

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3004

AVERAGE NUMBER OF UNITS IN SYS.= 2.274

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1207.853
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8282

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.27170000d+04
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.72

MAXIMUM LENGTH OF WAITING LINE = 12

AVERAGE TIME TO TRANSIT SYS. 0.75 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 521.11 HOURS..

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3011

AVERAGE NUMBER OF UNITS IN SYS.= 2.169

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1130.351
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8958

APPENDIX EE

OUTPUT LISTING
TRUCK SIMULATION PROGRAM
FORTRAN 4 VERSION WITH 6500 ELEMENT ARRAY

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.56700000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.71

MAXIMUM LENGTH OF WAITING LINE = 15

AVERAGE TIME TO TRANSIT SYS. 0.76 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00

SIMULATION RUN LENGTH 1053.03 HOURS..

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6006

AVERAGE NUMBER OF UNITS IN SYS.= 2.174

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2289.684
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8527

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.45900000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333

MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.72

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 0.70 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 1076.16 HOURS..

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6005

AVERAGE NUMBER OF UNITS IN SYS.= 1.946

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2094.400
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.7905

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.56100000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.70

MAXIMUM LENGTH OF WAITING LINE = 13

AVERAGE TIME TO TRANSIT SYS. 0.76 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 1083.88 HOURS..

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6005

AVERAGE NUMBER OF UNITS IN SYS.= 2.109

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2286.446
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.7706

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.66300000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.72

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 0.71 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 1068.28 HOURS..

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6005

AVERAGE NUMBER OF UNITS IN SYS.= 1.995

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2131.000
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8111

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.61300000d+03

```
 MEAN ARRIVAL TIME(MIAT) =          0.3333
 MEAN SERVICE TIME(MSVT) =          0.2500


 PROPORTION OF TIME DOCK CREW IS BUSY =     0.72

 MAXIMUM LENGTH OF WAITING LINE =       10

 AVERAGE TIME TO TRANSIT SYS.     0.70    HOURS..

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM  0.

 SIMULATION RUN LENGTH 1071.14    HOURS..

 NUMBER OF TRUCKS UNLOADED =     3000

 NUMBER OF RANDOM NUMBERS USED =       6002

 AVERAGE NUMBER OF UNITS IN SYS.=       1.954

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=    2093.365
(TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=         2.8017



     TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE.


 DSEED =           0.86700000d+03
 MEAN ARRIVAL TIME(MIAT) =          0.3333
 MEAN SERVICE TIME(MSVT) =          0.2500


 PROPORTION OF TIME DOCK CREW IS BUSY =     0.70

 MAXIMUM LENGTH OF WAITING LINE =       12

 AVERAGE TIME TO TRANSIT SYS.     0.71    HOURS..

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM  0.

 SIMULATION RUN LENGTH 1081.54    HOURS..

 NUMBER OF TRUCKS UNLOADED =     3000
```

NUMBER OF RANDOM NUMBERS USED = 6004

AVERAGE NUMBER OF UNITS IN SYS.= 1.982

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2143.365
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.7757

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.96900000d+03
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.70

MAXIMUM LENGTH OF WAITING LINE = 12

AVERAGE TIME TO TRANSIT SYS. 0.77 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00

SIMULATION RUN LENGTH 1072.07 HOURS..

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6005

AVERAGE NUMBER OF UNITS IN SYS.= 2.151

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2306.380
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8011

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

```
 DSEED =          0.10710000d+04
 MEAN ARRIVAL TIME(MIAT) =          0.3333
 MEAN SERVICE TIME(MSVT) =          0.2500


 PROPORTION OF TIME DOCK CREW IS BUSY =     0.70

 MAXIMUM LENGTH OF WAITING LINE =       12

 AVERAGE TIME TO TRANSIT SYS.     0.72    HOURS..

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM  0.

 SIMULATION RUN LENGTH 1078.29    HOURS..

 NUMBER OF TRUCKS UNLOADED =     3000

 NUMBER OF RANDOM NUMBERS USED =       6009

 AVERAGE NUMBER OF UNITS IN SYS.=       2.002

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=    2159.108
(TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=         2.7887



     TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE.


 DSEED =          0.11730000d+04
 MEAN ARRIVAL TIME(MIAT) =          0.3333
 MEAN SERVICE TIME(MSVT) =          0.2500


 PROPORTION OF TIME DOCK CREW IS BUSY =     0.70

 MAXIMUM LENGTH OF WAITING LINE =       12

 AVERAGE TIME TO TRANSIT SYS.     0.76    HOURS..

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM  0.

 SIMULATION RUN LENGTH 1082.48    HOURS..

 NUMBER OF TRUCKS UNLOADED =     3000
```

NUMBER OF RANDOM NUMBERS USED = 6006

AVERAGE NUMBER OF UNITS IN SYS.= 2.093

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2266.082
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.7751

TRUCK QUEUING PROBLEM:ANDERSON AND SWEENEY-SINGLE SERVER QUEUE.

DSEED = 0.27170000d+04
MEAN ARRIVAL TIME(MIAT) = 0.3333
MEAN SERVICE TIME(MSVT) = 0.2500

PROPORTION OF TIME DOCK CREW IS BUSY = 0.71

MAXIMUM LENGTH OF WAITING LINE = 12

AVERAGE TIME TO TRANSIT SYS. 0.76 HOURS..

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.

SIMULATION RUN LENGTH 1067.87 HOURS..

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6004

AVERAGE NUMBER OF UNITS IN SYS.= 2.122

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2265.961
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8112

APPENDIX FF

OUTPUT LISTING
TRUCK SIMULATION PROGRAM
ADA LINE-BY-LINE TRANSLATION WITH 3500 ELEMENT ARRAY
TELESOFT-ADA COMPILER VERSION 1.5

```
TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER
QUEUE


 DSEED=     5.6700000E+02
 MEAN ARRIVAL TIME(MIAT) =      3.3333334E-01
 MEAN SERVICE TIME(MSVT) =      2.5000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.3094196E-01

 MAXIMUM LENGTH OF WAITING LINE =     15

 AVERAGE TIME TO TRANSIT SYS. 8.0482091E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 6.6666669E-03

 SIMULATION RUN LENGTH 5.0935635E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =  1500

 NUMBER OF RANDOM NUMBERS USED =  3007

 AVERAGE NUMBER OF UNITS IN SYS.=     2.3701117E+00

 TOTAL NUMBER OF TRUCK HOURS IN TH' SYSTEM(S)= 1.2072314E+03
(TRU 'KS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.9547093E+00


     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=     4.5900001E+02
 MEAN ARRIVAL TIME(MIAT) =      3.3333334E-01
 MEAN SERVICE TIME(MSVT) =      2.5000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.3167362E-01
```

```
 MAXIMUM LENGTH OF WAITING LINE =     10

 AVERAGE TIME TO TRANSIT SYS. 7.0568180E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.0000000E+00

 SIMULATION RUN LENGTH 5.3324599E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =  1500

 NUMBER OF RANDOM NUMBERS USED =  3004

 AVERAGE NUMBER OF UNITS IN SYS.=     1.9850552E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.0585227E+03
(TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.8167114E+00


    TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      5.6100001E+02
 MEAN ARRIVAL TIME(MIAT) =       3.3333334E-01
 MEAN SERVICE TIME(MSVT) =       2.5000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.1636633E-01

 MAXIMUM LENGTH OF WAITING LINE =     13

 AVERAGE TIME TO TRANSIT SYS. 7.9491424E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.0000000E+00

 SIMULATION RUN LENGTH 5.2921595E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =  1500

 NUMBER OF RANDOM NUMBERS USED =  3008

 AVERAGE NUMBER OF UNITS IN SYS.=     2.2530903E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.1923713E+03
(TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.8457193E+00
```

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 6.6300001E+02
MEAN ARRIVAL TIME(MIAT) = 3.3333334E-01
MEAN SERVICE TIME(MSVT) = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.1621222E-01

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 6.9050850E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.0000000E+00

SIMULATION RUN LENGTH 5.3483600E+02HOURS.

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3004

AVERAGE NUMBER OF UNITS IN SYS.= 1.9365986E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.0357626E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8083374E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 6.1300001E+02
MEAN ARRIVAL TIME(MIAT) = 3.3333334E-01
MEAN SERVICE TIME(MSVT) = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.3744874E-01

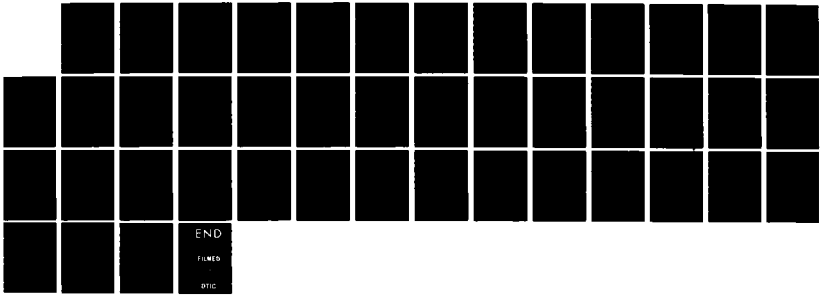
MAXIMUM LENGTH OF WAITING LINE = 10

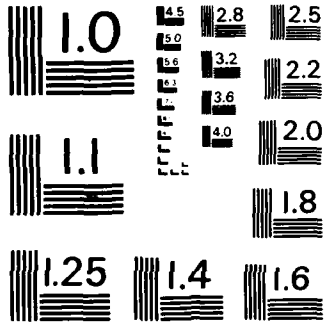
AVERAGE TIME TO TRANSIT SYS. 7.0562534E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.0000000E+00

AD-A161 715 AN ASSESSMENT OF ADA'S SUITABILITY IN GENERAL PURPOSE PROGRAMMING APPLICATIONS(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF SYST.. 3/3

UNCLASSIFIED L D CAVITT ET AL. SEP 85 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

SIMULATION RUN LENGTH 5.3041234E+02HOURS.

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3002

AVERAGE NUMBER OF UNITS IN SYS.= 1.9955001E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.0584379E+03
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8298738E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 8.6700000E+02
MEAN ARRIVAL TIME(MIAT) = 3.3333334E-01
MEAN SERVICE TIME(MSVT) = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.2206544E-01

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 6.9945359E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.0000000E+00

SIMULATION RUN LENGTH 5.2559552E+02HOURS.

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3004

AVERAGE NUMBER OF UNITS IN SYS.= 1.9961746E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.0491803E+03
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.8577108E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 9.6899995E+02

```
 MEAN ARRIVAL TIME(MIAT) =      3.3333334E-01
 MEAN SERVICE TIME(MSVT) =      2.5000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.2336530E-01

 MAXIMUM LENGTH OF WAITING LINE =     12

 AVERAGE TIME TO TRANSIT SYS. 7.9918785E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 4.0000000E-03

 SIMULATION RUN LENGTH 5.2588500E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =  1500

 NUMBER OF RANDOM NUMBERS USED =  3002

 AVERAGE NUMBER OF UNITS IN SYS.=     2.2795510E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.1987817E+03
(TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.8542361E+00


   TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      1.0709999E+03
 MEAN ARRIVAL TIME(MIAT) =      3.3333334E-01
 MEAN SERVICE TIME(MSVT) =      2.5000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.1392626E-01

 MAXIMUM LENGTH OF WAITING LINE =     10

 AVERAGE TIME TO TRANSIT SYS. 6.9153060E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.0000000E+00

 SIMULATION RUN LENGTH 5.3004603E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =  1500

 NUMBER OF RANDOM NUMBERS USED =  3002

 AVERAGE NUMBER OF UNITS IN SYS.=     1.9569921E+00
```

```
 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.0372959E+03
(TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.8318295E+00


     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      1.1729999E+03
 MEAN ARRIVAL TIME(MIAT) =      3.3333334E-01
 MEAN SERVICE TIME(MSVT) =      2.5000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.1630754E-01

 MAXIMUM LENGTH OF WAITING LINE =     12

 AVERAGE TIME TO TRANSIT SYS. 8.0523452E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.0000000E+00

 SIMULATION RUN LENGTH 5.3108630E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =  1500

 NUMBER OF RANDOM NUMBERS USED =  3004

 AVERAGE NUMBER OF UNITS IN SYS.=     2.2743041E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.2078517E+03
(TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.8281655E+00


     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      2.7170000E+03
 MEAN ARRIVAL TIME(MIAT) =      3.3333334E-01
 MEAN SERVICE TIME(MSVT) =      2.5000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.2412753E-01

 MAXIMUM LENGTH OF WAITING LINE =     12

 AVERAGE TIME TO TRANSIT SYS. 7.5356698E-01HOURS.
```

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.0000000E+00

SIMULATION RUN LENGTH 5.2110710E+02HOURS.

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3011

AVERAGE NUMBER OF UNITS IN SYS.= 2.1691327E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.1303504E+03
(TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS!PER HR= 2.8957579E+00

APPENDIX GG

OUTPUT LISTING
TRUCK SIMULATION PROGRAM
ADA LINE-BY-LINE TRANSLATION WITH 3500 ELEMENT ARRAY
VADS COMPILER RELEASE V04.06

```
 TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER
QUEUE


 DSEED=      5.67000000E+02
 MEAN ARRIVAL TIME(MIAT) =       3.33333333E-01
 MEAN SERVICE TIME(MSVT) =       2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.30942508E-01

 MAXIMUM LENGTH OF WAITING LINE =          15

 AVERAGE TIME TO TRANSIT SYS. 8.04808941E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 6.66666667E-03

 SIMULATION RUN LENGTH 5.09356313E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =        1500

 NUMBER OF RANDOM NUMBERS USED =        3007

 AVERAGE NUMBER OF UNITS IN SYS.=     2.37007647E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
1.20721341E+03     (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.95470962E+00


     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      4.59000000E+02
 MEAN ARRIVAL TIME(MIAT) =       3.33333333E-01
 MEAN SERVICE TIME(MSVT) =       2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.31673147E-01
```

```
 MAXIMUM LENGTH OF WAITING LINE =          10

 AVERAGE TIME TO TRANSIT SYS. 7.05674893E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.00000000E+00

 SIMULATION RUN LENGTH 5.33246154E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =        1500

 NUMBER OF RANDOM NUMBERS USED =        3004

 AVERAGE NUMBER OF UNITS IN SYS.=     1.98503511E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
1.05851234E+03     (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.81671042E+00


     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      5.61000000E+02
 MEAN ARRIVAL TIME(MIAT) =      3.33333333E-01
 MEAN SERVICE TIME(MSVT) =      2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.16366562E-01

 MAXIMUM LENGTH OF WAITING LINE =          13

 AVERAGE TIME TO TRANSIT SYS. 7.94906046E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.00000000E+00

 SIMULATION RUN LENGTH 5.29215984E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =        1500

 NUMBER OF RANDOM NUMBERS USED =        3008

 AVERAGE NUMBER OF UNITS IN SYS.=     2.25306700E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
1.19235907E+03     (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.84571904E+00
```

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 6.63000000E+02
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.16212171E-01

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 6.90500838E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 5.34835834E+02HOURS.

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3004

AVERAGE NUMBER OF UNITS IN SYS.= 1.93657790E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.03575126E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.80833838E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 6.13000000E+02
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.37447834E-01

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 7.05614019E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 5.30412527E+02HOURS.

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3002

AVERAGE NUMBER OF UNITS IN SYS.= 1.99546763E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
1.05842103E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.82987283E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 8.67000000E+02
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.22066315E-01

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 6.99446403E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 5.25595120E+02HOURS.

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3004

AVERAGE NUMBER OF UNITS IN SYS.= 1.99615553E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
1.04916960E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.85771298E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 9.69000000E+02

```
 MEAN ARRIVAL TIME(MIAT) =      3.33333333E-01
 MEAN SERVICE TIME(MSVT) =      2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.23365362E-01

 MAXIMUM LENGTH OF WAITING LINE =          12

 AVERAGE TIME TO TRANSIT SYS. 7.99180875E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 4.00000000E-03

 SIMULATION RUN LENGTH 5.25884911E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =        1500

 NUMBER OF RANDOM NUMBERS USED =        3002

 AVERAGE NUMBER OF UNITS IN SYS.=     2.27953168E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
1.19877131E+03     (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.85423668E+00



     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      1.07100000E+03
 MEAN ARRIVAL TIME(MIAT) =      3.33333333E-01
 MEAN SERVICE TIME(MSVT) =      2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.13927243E-01

 MAXIMUM LENGTH OF WAITING LINE =          10

 AVERAGE TIME TO TRANSIT SYS. 6.91520862E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.00000000E+00

 SIMULATION RUN LENGTH 5.30045391E+02HOURS.

 NUMBER OF TRUCKS UNLOADED =        1500

 NUMBER OF RANDOM NUMBERS USED =        3002

 AVERAGE NUMBER OF UNITS IN SYS.=     1.95696691E+00
```

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
1.03723129E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.83183294E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 1.17300000E+03
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.16307133E-01

MAXIMUM LENGTH OF WAITING LINE = 12

AVERAGE TIME TO TRANSIT SYS. 8.05223633E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 5.31086518E+02HOURS.

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3004

AVERAGE NUMBER OF UNITS IN SYS.= 2.27427247E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
1.20783544E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.82816443E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 2.71700000E+03
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.24128190E-01

MAXIMUM LENGTH OF WAITING LINE = 12

AVERAGE TIME TO TRANSIT SYS. 7.53558208E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 5.21107093E+02HOURS.

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3011

AVERAGE NUMBER OF UNITS IN SYS.= 2.16910752E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 1.13033731E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.89575794E+00

APPENDIX HH

OUTPUT LISTING
TRUCK SIMULATION PROGRAM
ADA LINE-BY-LINE TRANSLATION WITH 6500 ELEMENT ARRAY
VADS COMPILER RELEASE V04.06

```
 TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER
QUEUE


 DSEED=      5.67000000E+02
 MEAN ARRIVAL TIME(MIAT) =       3.33333333E-01
 MEAN SERVICE TIME(MSVT) =       2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.03899000E-01

 MAXIMUM LENGTH OF WAITING LINE =          15

 AVERAGE TIME TO TRANSIT SYS. 7.63210812E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 3.33333333E-03

 SIMULATION RUN LENGTH 1.05303208E+03HOURS.

 NUMBER OF TRUCKS UNLOADED =        3000

 NUMBER OF RANDOM NUMBERS USED =        6006

 AVERAGE NUMBER OF UNITS IN SYS.=     2.17432353E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
2.28963244E+03     (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.85271461E+00


     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      4.59000000E+02
 MEAN ARRIVAL TIME(MIAT) =       3.33333333E-01
 MEAN SERVICE TIME(MSVT) =       2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.21658101E-01
```

```
 MAXIMUM LENGTH OF WAITING LINE =          10

 AVERAGE TIME TO TRANSIT SYS. 6.98115739E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.00000000E+00

 SIMULATION RUN LENGTH 1.07615362E+03HOURS.

 NUMBER OF TRUCKS UNLOADED =        3000

 NUMBER OF RANDOM NUMBERS USED =        6005

 AVERAGE NUMBER OF UNITS IN SYS.=     1.94614150E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
2.09434722E+03     (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.79049379E+00


     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      5.61000000E+02
 MEAN ARRIVAL TIME(MIAT) =      3.33333333E-01
 MEAN SERVICE TIME(MSVT) =      2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.01955668E-01

 MAXIMUM LENGTH OF WAITING LINE =          13

 AVERAGE TIME TO TRANSIT SYS. 7.62128362E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.00000000E+00

 SIMULATION RUN LENGTH 1.08388089E+03HOURS.

 NUMBER OF TRUCKS UNLOADED =        3000

 NUMBER OF RANDOM NUMBERS USED =        6005

 AVERAGE NUMBER OF UNITS IN SYS.=     2.10944312E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
2.28638509E+03     (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=      2.77059964E+00
```

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 6.63000000E+02
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.23701229E-01

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 7.10316696E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 1.06827350E+03HOURS.

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6005

AVERAGE NUMBER OF UNITS IN SYS.= 1.99476078E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2.13095009E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.81107787E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 6.13000000E+02
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.24916698E-01

MAXIMUM LENGTH OF WAITING LINE = 10

AVERAGE TIME TO TRANSIT SYS. 6.97767120E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 1.07114171E+03HOURS.

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6002

AVERAGE NUMBER OF UNITS IN SYS.= 1.95427116E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
2.09330136E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.80168344E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 8.67000000E+02
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.01069610E-01

MAXIMUM LENGTH OF WAITING LINE = 12

AVERAGE TIME TO TRANSIT SYS. 7.14437305E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 1.08154439E+03HOURS.

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6004

AVERAGE NUMBER OF UNITS IN SYS.= 1.98171425E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
2.14331192E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.77566047E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 9.69000000E+02

```
 MEAN ARRIVAL TIME(MIAT) =      3.33333333E-01
 MEAN SERVICE TIME(MSVT) =      2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.04902495E-01

 MAXIMUM LENGTH OF WAITING LINE =          12

 AVERAGE TIME TO TRANSIT SYS. 7.68772879E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 2.00000000E-03

 SIMULATION RUN LENGTH 1.07207391E+03HOURS.

 NUMBER OF TRUCKS UNLOADED =        3000

 NUMBER OF RANDOM NUMBERS USED =        6005

 AVERAGE NUMBER OF UNITS IN SYS.=    2.15126832E+00

 TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
2.30631864E+03     (TRUCKS PER HR)

 AVERAGE NUMBER OF ARRIVALS PER HR=     2.80111284E+00


     TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE
SERVER QUEUE


 DSEED=      1.07100000E+03
 MEAN ARRIVAL TIME(MIAT) =      3.33333333E-01
 MEAN SERVICE TIME(MSVT) =      2.50000000E-01

 PROPORTION OF TIME DOCK CREW IS BUSY = 7.03083724E-01

 MAXIMUM LENGTH OF WAITING LINE =          12

 AVERAGE TIME TO TRANSIT SYS. 7.19684274E-01HOURS.

 PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE
SYSTEM 0.00000000E+00

 SIMULATION RUN LENGTH 1.07828782E+03HOURS.

 NUMBER OF TRUCKS UNLOADED =        3000

 NUMBER OF RANDOM NUMBERS USED =        6009

 AVERAGE NUMBER OF UNITS IN SYS.=    2.00229733E+00
```

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
2.15905282E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.78868030E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 1.17300000E+03
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 6.99513375E-01

MAXIMUM LENGTH OF WAITING LINE = 12

AVERAGE TIME TO TRANSIT SYS. 7.55341685E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 1.08247663E+03HOURS.

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6006

AVERAGE NUMBER OF UNITS IN SYS.= 2.09337088E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)=
2.26602506E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.77511766E+00

TRUCK QUEUING PROBLEM: ANDERSON AND SWEENEY-SINGLE SERVER QUEUE

DSEED= 2.71700000E+03
MEAN ARRIVAL TIME(MIAT) = 3.33333333E-01
MEAN SERVICE TIME(MSVT) = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.10936320E-01

MAXIMUM LENGTH OF WAITING LINE = 12

AVERAGE TIME TO TRANSIT SYS. 7.55301701E-01HOURS.

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS.. IN THE SYSTEM 0.00000000E+00

SIMULATION RUN LENGTH 1.06786648E+03HOURS.

NUMBER OF TRUCKS UNLOADED = 3000

NUMBER OF RANDOM NUMBERS USED = 6004

AVERAGE NUMBER OF UNITS IN SYS.= 2.12189927E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S)= 2.26590510E+03 (TRUCKS PER HR)

AVERAGE NUMBER OF ARRIVALS PER HR= 2.81121287E+00

APPENDIX II

OUTPUT LISTING
TRUCK SIMULATION PROGRAM
ADA REDESIGN
TELESOFT-ADA COMPILER VERSION 1.5

```
RANDOM NUMBER GENERATOR SEED 5.6700000E+02
MEAN INTERARRIVAL TIME =  3.3333334E-01
MEAN SERVICE TIME =  2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY =  7.3094196E-01

MAXIMUM LENGTH OF WAITING LINE =      16

AVERAGE TIME TO TRANSIT SYSTEM =  8.0482091E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS = 6.6666669E-
03

SIMULATION RUN LENGTH =  5.0935635E+02  HOURS

NUMBER OF TRUCKS UNLOADED =    1500

NUMBER OF RANDOM NUMBERS USED =    3006

AVERAGE NUMBER OF UNITS IN SYS. =  2.3701117E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.2072314E+03

AVERAGE NUMBER OF ARRIVALS PER HR =  2.9527462E+00


RANDOM NUMBER GENERATOR SEED 4.5900001E+02
MEAN INTERARRIVAL TIME =  3.3333334E-01
MEAN SERVICE TIME =  2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY =  7.3167362E-01

MAXIMUM LENGTH OF WAITING LINE =      11

AVERAGE TIME TO TRANSIT SYSTEM =  7.0568180E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.0000000E+00

SIMULATION RUN LENGTH =  5.3324599E+02  HOURS
```

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3003

AVERAGE NUMBER OF UNITS IN SYS. = 1.9850552E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.0585227E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.8148360E+00

RANDOM NUMBER GENERATOR SEED 5.6100001E+02
MEAN INTERARRIVAL TIME = 3.3333334E-01
MEAN SERVICE TIME = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.1636633E-01

MAXIMUM LENGTH OF WAITING LINE = 14

AVERAGE TIME TO TRANSIT SYSTEM = 7.9491424E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.0000000E+00

SIMULATION RUN LENGTH = 5.2921595E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3007

AVERAGE NUMBER OF UNITS IN SYS. = 2.2530903E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.1923713E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.8438296E+00

RANDOM NUMBER GENERATOR SEED 6.6300001E+02
MEAN INTERARRIVAL TIME = 3.3333334E-01
MEAN SERVICE TIME = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.1621222E-01

MAXIMUM LENGTH OF WAITING LINE = 11

AVERAGE TIME TO TRANSIT SYSTEM = 6.9050850E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.0000000E+00

SIMULATION RUN LENGTH = 5.3483600E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3003

AVERAGE NUMBER OF UNITS IN SYS. = 1.9365986E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.0357626E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.8064678E+00

RANDOM NUMBER GENERATOR SEED 6.1300001E+02
MEAN INTERARRIVAL TIME = 3.3333334E-01
MEAN SERVICE TIME = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.3744874E-01

MAXIMUM LENGTH OF WAITING LINE = 11

AVERAGE TIME TO TRANSIT SYSTEM = 7.0562534E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.0000000E+00

SIMULATION RUN LENGTH = 5.3041234E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3001

AVERAGE NUMBER OF UNITS IN SYS. = 1.9955001E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.0584379E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.8279884E+00

RANDOM NUMBER GENERATOR SEED 8.6700000E+02
MEAN INTERARRIVAL TIME = 3.3333334E-01
MEAN SERVICE TIME = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.2206544E-01

```
MAXIMUM LENGTH OF WAITING LINE =      11

AVERAGE TIME TO TRANSIT SYSTEM =  6.9945359E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.0000000E+00

SIMULATION RUN LENGTH =  5.2559552E+02  HOURS

NUMBER OF TRUCKS UNLOADED =   1500

NUMBER OF RANDOM NUMBERS USED =   3003

AVERAGE NUMBER OF UNITS IN SYS. =  1.9961746E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.0491803E+03

AVERAGE NUMBER OF ARRIVALS PER HR =  2.8558082E+00


RANDOM NUMBER GENERATOR SEED 9.6899995E+02
MEAN INTERARRIVAL TIME =  3.3333334E-01
MEAN SERVICE TIME =  2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY =  7.2336530E-01

MAXIMUM LENGTH OF WAITING LINE =      13

AVERAGE TIME TO TRANSIT SYSTEM =  7.9918785E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS = 4.0000000E-
03

SIMULATION RUN LENGTH =  5.2588500E+02  HOURS

NUMBER OF TRUCKS UNLOADED =   1500

NUMBER OF RANDOM NUMBERS USED =   3001

AVERAGE NUMBER OF UNITS IN SYS. =  2.2795510E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.1987817E+03

AVERAGE NUMBER OF ARRIVALS PER HR =  2.8523344E+00


RANDOM NUMBER GENERATOR SEED 1.0709999E+03
MEAN INTERARRIVAL TIME =  3.3333334E-01
```

MEAN SERVICE TIME = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.1392626E-01

MAXIMUM LENGTH OF WAITING LINE = 11

AVERAGE TIME TO TRANSIT SYSTEM = 6.9153060E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS = 0.0000000E+00

SIMULATION RUN LENGTH = 5.3004603E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3001

AVERAGE NUMBER OF UNITS IN SYS. = 1.9569921E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) = 1.0372959E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.8299429E+00

RANDOM NUMBER GENERATOR SEED 1.1729999E+03
MEAN INTERARRIVAL TIME = 3.3333334E-01
MEAN SERVICE TIME = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.1630754E-01

MAXIMUM LENGTH OF WAITING LINE = 13

AVERAGE TIME TO TRANSIT SYSTEM = 8.0523452E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS = 0.0000000E+00

SIMULATION RUN LENGTH = 5.3108630E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3003

AVERAGE NUMBER OF UNITS IN SYS. = 2.2743041E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) = 1.2078517E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.8262827E+00

RANDOM NUMBER GENERATOR SEED 2.7170000E+03
MEAN INTERARRIVAL TIME = 3.3333334E-01
MEAN SERVICE TIME = 2.5000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.2412753E-01

MAXIMUM LENGTH OF WAITING LINE = 13

AVERAGE TIME TO TRANSIT SYSTEM = 7.5356698E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.0000000E+00

SIMULATION RUN LENGTH = 5.2110710E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3010

AVERAGE NUMBER OF UNITS IN SYS. = 2.1691327E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.1303504E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.8938388E+00

APPENDIX JJ

OUTPUT LISTING
TRUCK SIMULATION PROGRAM
ADA REDESIGN
VADS COMPILER RELEASE V04.06

```
RANDOM NUMBER GENERATOR SEED 5.67J00000E+02
MEAN INTERARRIVAL TIME =  3.33333333E-01
MEAN SERVICE TIME =  2.50000J00E-01

PROPORTION OF TIME DOCK CREW IS BUSY =  7.30942508E-01

MAXIMUM LENGTH OF WAITING LINE =          16

AVERAGE TIME TO TRANSIT SYSTEM =  8.04308941E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
6.66666667E-03

SIMULATION RUN LENGTH =  5.09356313E+02  HOURS

NUMBER OF TRUCKS UNLOADED =        1500

NUMBER OF RANDOM NUMBERS USED =        3006

AVERAGE NUMBER OF UNITS IN SYS. =  2.37007647E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.20721341E+03

AVERAGE NUMBER OF ARRIVALS PER HR =  2.95274636E+00


RANDOM NUMBER GENERATOR SEED 4.59000000E+02
MEAN INTERARRIVAL TIME =  3.33333333E-01
MEAN SERVICE TIME =  2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY =  7.31673147E-01

MAXIMUM LENGTH OF WAITING LINE =          11

AVERAGE TIME TO TRANSIT SYSTEM =  7.05674893E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.00000000E+00
```

```
SIMULATION RUN LENGTH =  5.33246154E+02  HOURS

NUMBER OF TRUCKS UNLOADED =         1500

NUMBER OF RANDOM NUMBERS USED =         3003

AVERAGE NUMBER OF UNITS IN SYS. =  1.98503511E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.05851234E+03

AVERAGE NUMBER OF ARRIVALS PER HR =  2.81483512E+00


RANDOM NUMBER GENERATOR SEED 5.61000000E+02
MEAN INTERARRIVAL TIME =  3.33333333E-01
MEAN SERVICE TIME =  2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY =  7.16366562E-01

MAXIMUM LENGTH OF WAITING LINE =           14

AVERAGE TIME TO TRANSIT SYSTEM =  7.94906046E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.00000000E+00

SIMULATION RUN LENGTH =  5.29215984E+02  HOURS

NUMBER OF TRUCKS UNLOADED =         1500

NUMBER OF RANDOM NUMBERS USED =         3007

AVERAGE NUMBER OF UNITS IN SYS. =  2.25306700E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.19235907E+03

AVERAGE NUMBER OF ARRIVALS PER HR =  2.84382945E+00


RANDOM NUMBER GENERATOR SEED 6.63000000E+02
MEAN INTERARRIVAL TIME =  3.33333333E-01
MEAN SERVICE TIME =  2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY =  7.16212171E-01

MAXIMUM LENGTH OF WAITING LINE =           11
```

AVERAGE TIME TO TRANSIT SYSTEM = 6.90500838E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS = 0.00000000E+00

SIMULATION RUN LENGTH = 5.34835834E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3003

AVERAGE NUMBER OF UNITS IN SYS. = 1.93657790E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) = 1.03575126E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.80646865E+00

RANDOM NUMBER GENERATOR SEED 6.13000000E+02
MEAN INTERARRIVAL TIME = 3.33333333E-01
MEAN SERVICE TIME = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.37447834E-01

MAXIMUM LENGTH OF WAITING LINE = 11

AVERAGE TIME TO TRANSIT SYSTEM = 7.05614019E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS = 0.00000000E+00

SIMULATION RUN LENGTH = 5.30412527E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3001

AVERAGE NUMBER OF UNITS IN SYS. = 1.99546763E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) = 1.05842103E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.82798751E+00

RANDOM NUMBER GENERATOR SEED 8.67000000E+02
MEAN INTERARRIVAL TIME = 3.33333333E-01
MEAN SERVICE TIME = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.22066315E-01

MAXIMUM LENGTH OF WAITING LINE = 11

AVERAGE TIME TO TRANSIT SYSTEM = 6.99446403E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS = 0.00000000E+00

SIMULATION RUN LENGTH = 5.25595120E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3003

AVERAGE NUMBER OF UNITS IN SYS. = 1.99615553E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) = 1.04916960E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.85581038E+00

RANDOM NUMBER GENERATOR SEED 9.69000000E+02
MEAN INTERARRIVAL TIME = 3.33333333E-01
MEAN SERVICE TIME = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.23365362E-01

MAXIMUM LENGTH OF WAITING LINE = 13

AVERAGE TIME TO TRANSIT SYSTEM = 7.99180875E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS = 4.00000000E-03

SIMULATION RUN LENGTH = 5.25884911E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3001

AVERAGE NUMBER OF UNITS IN SYS. = 2.27953168E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) = 1.19877131E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.85233512E+00

```
RANDOM NUMBER GENERATOR SEED 1.07100000E+03
MEAN INTERARRIVAL TIME =  3.33333333E-01
MEAN SERVICE TIME =  2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY =  7.13927243E-01

MAXIMUM LENGTH OF WAITING LINE =          11

AVERAGE TIME TO TRANSIT SYSTEM =  6.91520862E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.00000000E+00

SIMULATION RUN LENGTH =  5.30045391E+02  HOURS

NUMBER OF TRUCKS UNLOADED =        1500

NUMBER OF RANDOM NUMBERS USED =        3001

AVERAGE NUMBER OF UNITS IN SYS. =  1.95696691E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.03728129E+03

AVERAGE NUMBER OF ARRIVALS PER HR =  2.82994631E+00


RANDOM NUMBER GENERATOR SEED 1.17300000E+03
MEAN INTERARRIVAL TIME =  3.33333333E-01
MEAN SERVICE TIME =  2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY =  7.16307133E-01

MAXIMUM LENGTH OF WAITING LINE =          13

AVERAGE TIME TO TRANSIT SYSTEM =  8.05223630E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.00000000E+00

SIMULATION RUN LENGTH =  5.31086518E+02  HOURS

NUMBER OF TRUCKS UNLOADED =        1500

NUMBER OF RANDOM NUMBERS USED =        3003

AVERAGE NUMBER OF UNITS IN SYS. =  2.27427247E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.20783544E+03

AVERAGE NUMBER OF ARRIVALS PER HR =  2.82628150E+00
```

RANDOM NUMBER GENERATOR SEED 2.71700000E+03
MEAN INTERARRIVAL TIME = 3.33333333E-01
MEAN SERVICE TIME = 2.50000000E-01

PROPORTION OF TIME DOCK CREW IS BUSY = 7.24128190E-01

MAXIMUM LENGTH OF WAITING LINE = 13

AVERAGE TIME TO TRANSIT SYSTEM = 7.53558208E-01

PROPORTION OF TRUCKS TAKING FOUR OR MORE HOURS =
0.00000000E+00

SIMULATION RUN LENGTH = 5.21107093E+02 HOURS

NUMBER OF TRUCKS UNLOADED = 1500

NUMBER OF RANDOM NUMBERS USED = 3010

AVERAGE NUMBER OF UNITS IN SYS. = 2.16910752E+00

TOTAL NUMBER OF TRUCK HOURS IN THE SYSTEM(S) =
1.13033731E+03

AVERAGE NUMBER OF ARRIVALS PER HR = 2.89383895E+00

APPENDIX KK

OUTPUT LISTING
LIBRARY MAINTENANCE PROGRAM
ORIGINAL PASCAL VERSION

```
% liblist
WOULD YOU LIKE TO INSERT OR DELETE A BOOK OR VIEW THE FILE?
TYPE I OR D OR V: V

ANNA KARENINA
TOLSTOY, COUNT LEO
        10

END OF LIBRARY FILE

TYPE I TO INSERT, D TO DELETE, OR V TO VIEW FILE, OR F TO
FINISH:I

TYPE THE NAME OF THE BOOK:
WAR AND PEACE

TYPE THE NAME OF THE AUTHOR:
TOLSTOY, COUNTLEO

TYPE THE CALL NUMBER OF THE BOOK:
5

TYPE I TO INSERT, D TO DELETE, OR V TO VIEW FILE, OR F TO
FINISH:V

WAR AND PEACE
TOLSTOY, COUNT LEO
         5
ANNA KARENINA
TOLSTOY, COUNT LEO
        10

END OF LIBRARY FILE

TYPE I TO INSERT, D TO DELETE, OR V TO VIEW FILE, OR F TO
FINISH:I

TYPE THE NAME OF THE BOOK:
JOE MAMA

TYPE THE NAME OF THE AUTHOR:
JOE DADDY
```

```
TYPE THE CALL NUMBER OF THE BOOK:
7

TYPE I TO INSERT, D TO DELETE, OR V TO VIEW FILE, OR F TO
FINISH:I

TYPE THE NAME OF THE BOOK:
JOE DADDY

TYPE THE NAME OF THE AUTHOR:
JOE MAMA

TYPE THE CALL NUMBER OF THE BOOK:
15

TYPE I TO INSERT, D TO DELETE, OR V TO VIEW FILE, OR F TO
FINISH:V

WAR AND PEACE
TOLSTOY, COUNT LEO
         5
JOE MAMA
JOE DADDY
         7
ANNA KARENINA
TOLSTOY, COUNT LEO
        10
JOE DADDY
JOE MAMA
        15

END OF LIBRARY FILE

TYPE I TO INSERT, D TO DELETE, OR V TO VIEW FILE, OR F TO
FINISH:D

TYPE THE CALL NUMBER OF THE BOOK:10


TYPE I TO INSERT, D TO DELETE, OR V TO VIEW FILE, OR F TO
FINISH:V

WAR AND PEACE
TOLSTOY, COUNT LEO
         5
JOE MAMA
JOE DADDY
         7
JOE DADDY
JOE MAMA
        15
```

```
END OF LIBRARY FILE

TYPE I TO INSERT, D TO DELETE, OR V TO VIEW FILE, OR F TO
FINISH:F

LIBRARY FILE IS NOW UPDATED


%
```

APPENDIX LL

OUTPUT LISTING
LIBRARY MAINTENANCE PROGRAM
ADA LINE-BY-LINE TRANSLATION
VADS COMPILER RELEASE V04.06

```
% lib
WOULD YOU LIKE TO INSERT OR DELETE A BOOK OR VIEW THE FILE?
TYPE I OR D OR V: V
ANNA KARENINA
TOLSTOY, COUNT LEO
          10

END OF LIBRARY FILE


TYPE I TO INSERT, D TO DELETE, V TO VIEW FILE OR F TO
FINISH: I

TYPE THE NAME OF THE BOOK:
WAR AND PEACE

TYPE THE NAME OF THE AUTHOR:
TOLSTOY, COUNT LEO

TYPE THE CALLNO OF THE BOOK:
5

TYPE I TO INSERT, D TO DELETE, V TO VIEW FILE OR F TO
FINISH: V

WAR AND PEACE
TOLSTOY, COUNT LEO
           5
ANNA KARENINA
TOLSTOY, COUNT LEO
          10

END OF LIBRARY FILE


TYPE I TO INSERT, D TO DELETE, V TO VIEW FILE OR F TO
FINISH: I

TYPE THE NAME OF THE BOOK:
JOE MAMA
```

```
TYPE THE NAME OF THE AUTHOR:
JOE DADDY

TYPE THE CALLNO OF THE BOOK:
7

TYPE I TO INSERT, D TO DELETE, V TO VIEW FILE OR F TO
FINISH: I

TYPE THE NAME OF THE BOOK:
JOE DADDY

TYPE THE NAME OF THE AUTHOR:
JOE MAMA

TYPE THE CALLNO OF THE BOOK:
15

TYPE I TO INSERT, D TO DELETE, V TO VIEW FILE OR F TO
FINISH: V

WAR AND PEACE
TOLSTOY, COUNT LEO
          5
JOE MAMA
JOE DADDY
          7
ANNA KARENINA
TOLSTOY, COUNT LEO
         10
JOE DADDY
JOE MAMA
         15

END OF LIBRARY FILE


TYPE I TO INSERT, D TO DELETE, V TO VIEW FILE OR F TO
FINISH: D

TYPE THE CALL NUMBER OF THE BOOK:10

TYPE I TO INSERT, D TO DELETE, V TO VIEW FILE OR F TO
FINISH: V

WAR AND PEACE
TOLSTOY, COUNT LEO
          5
JOE MAMA
JOE DADDY
          7
JOE DADDY
```

```
JOE MAMA
        15

END OF LIBRARY FILE


TYPE I TO INSERT, D TO DELETE, V TO VIEW FILE OR F TO
FINISH: F

LIBRARY FILE IS NOW UPDATED

%
```

APPENDIX MM

OUTPUT LISTING
LIBRARY MAINTENANCE PROGRAM
ADA REDESIGN
VADS COMPILER RELEASE V04.06

```
% lib_main

LIBRARY MAINTENANCE PROGRAM


What do you want to do?
    'I' = insert a book
    'D' = delete a book
    'P' = print library list
    'Q' = quit
P

          10  ANNA KARENINA          by TOLSTOY, COUNT LEO

What do you want to do?
    'I' = insert a book
    'D' = delete a book
    'P' = print library list
    'Q' = quit
I

Enter book title
WAR AND PEACE

Enter author name
TOLSTOY, COUNT LEO

Enter call number
5

What do you want to do?
    'I' = insert a book
    'D' = delete a book
    'P' = print library list
    'Q' = quit
P

           5  WAR AND PEACE          by TOLSTOY, COUNT LEO
```

```
         10  ANNA KARENINA          by TOLSTOY, COUNT LEO

What do you want to do?
   'I' = insert a book
   'D' = delete a book
   'P' = print library list
   'Q' = quit
I

Enter book title
JOE MAMA

Enter author name
JOE DADDY

Enter call number
7

What do you want to do?
   'I' = insert a book
   'D' = delete a book
   'P' = print library list
   'Q' = quit
I

Enter book title
JOE DADDY

Enter autnor name
JOE MAMA

Enter call number
15

What do you want to do?
   'I' = insert a book
   'D' = delete a book
   'P' = print library list
   'Q' = quit
P

          5  WAR AND PEACE          by TOLSTOY, COUNT LEO
          7  JOE MAMA               by JOE DADDY
         10  ANNA KARENINA          by TOLSTOY, COUNT LEO
         15  JOE DADDY              by JOE MAMA

What do you want to do?
   'I' = insert a book
   'D' = delete a book
   'P' = print library list
   'Q' = quit
D

Enter call number of book to be deleted.
```

```
10

What do you want to do?
   'I' = insert a book
   'D' = delete a book
   'P' = print library list
   'Q' = quit
P

          5  WAR AND PEACE          by TOLSTOY, COUNT LEO
          7  JOE MAMA               by JOE DADDY
         15  JOE DADDY              by JOE MAMA


What do you want to do?
   'I' = insert a book
   'D' = delete a book
   'P' = print library list
   'Q' = quit
Q
```

BIBLIOGRAPHY

1. Banks, Jerry and John S. Carson II. Discrete-Event System Simulation. Englewood Cliffs NJ: Prentice-Hall Inc., 1984.

2. Booch, Grady. Software Engineering with Ada. Menlo Park: The Benjamin/Cummings Publisning Company, 1983.

3. Broad, William J. "Pentagon Orders End to Computer Babel," Science, 211: 31-33 (2 January 1981).

4. Coar, David. "Pascal, Ada, and Modula-2," Byte, 9: 215-232 (August 1984).

5. Crafts, Ralph E. "Ada for Business and Other Non-DoD Applications," Proceedings of the Annual Conference on Ada (Trademark) Technology (2nd) Held at Hampton VA, 70-73 (27-23 March 1984) (AD-P003 425).

6. DaCosta, Robert. "Ada: An Indepth Look," Defense Science and Electronics, 3: 38-73 (March 1984).

7. Department of Defense. Military Standard Ada Programming Languagae. Washington DC, ANSI/MIL-STD-1815A (January 1983).

8. Enrenfried, D.H. Feasibility Assessment of Jovial to Ada Translation. Technical Paper, Air Force Wright Aeronautical Labs Wright-Patterson AFB OH. August 1983 (AD-A134 857).

9. Emory, William C. Business Research Methods. Homewood IL: Richard D. Irwin Inc., 1980.

10. Fawcette, James E. "Ada Goes to Work," Defense Electronics, 14: 60-81 (July 1982).

11. Fisher, David A. "DoD's Common Programming Language Effort," Computer, 3: 24-33 (March 1978).

12. Fonash, Peter. "Ada - Program Overview," Signal, 37: 27-31 (July 1983).

13. -----. "Parlez-Vous 'Ada'?" Program Manager, 12: 5-10 (July-August 1983).

14. LeBlanc, Richard J. and John L. Goode. "Ada and Software Development: A New Concept in Language Design," Computer, 15: 75-82 (May 1983).

15. Miller, Alan R. FORTRAN Programs for Scientists and Engineers. Berkeley CA: Sybex Inc., 1982.

16. Pyle, I.C. The Ada Program Language. Englewood Cliffs NJ: Prentice Hall International, 1981.

17. Schmitz, Gregory H. "Can Ada Lower the Cost of Software in C3I Systems?" Signal, 37: 75-77 (August 1983).

18. Sherman, Bruce. "Design of the First Ada KAPSE Interface," Defense Electronics, 16: 141-149 (April 1984).

19. U.S. Department of the Air Force, Directorate of Integration and Technology. Data Project Directive HAF-P83-006, Washington DC (23 December 1983).

20. Wylie, George T. Lt Cmdr and Watt, Thomas R. Lt, Utilization of Ada as a Program Design Language. MS thesis, Naval Post Graduate School, Monterey CA, Jun 1983 (AD-A132 244).

21. Wichman, B.A. "A Comparison of Pascal and Ada," The Computer Journal, 25: 248-252 (May 1982).

22. Zaks, Rodnay. Introduction To Pascal Including UCSD Pascal (Second Edition). Berkeley CA: Sybex Inc., 1981.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A161715

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GLM/LSM/ 85S-62	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Systems and Logistics 6b. OFFICE SYMBOL (If applicable) AFIT/LSM	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) See Box 19.	
12. PERSONAL AUTHOR(S) Larry D. Cavitt, Capt, USAF Anthony A. Panek, Capt, USAF	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1985 September 15. PAGE COUNT 234
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Ada, Programming Language, Software Readability
Software Maintenance, Simulation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: AN ASSESSMENT OF ADA'S SUITABILITY IN GENERAL PURPOSE PROGRAMMING APPLICATIONS

Thesis Chairman: Patricia K. Lawlis, Captain, USAF
Instructor in Mathematics and Computer Science

Approved for public release: IAW AFR 190-1.
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AFSC)
Wright-Patterson AFB OH 45433
11 Sept 85

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Patricia K. Lawlis, Captain, USAF	22b. TELEPHONE NUMBER (Include Area Code) 513-255-3098 22c. OFFICE SYMBOL AFIT/ENC

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

The Ada programming language is the result of a multiyear effort under the sponsorship of the Department of Defense (DoD) to obtain the benifits of a single DoD-wide language for use in embedded computer systems. The language was developed to reduce or eliminate many of the serious and costly problems associated with the development and maintenance of software for embedded systems. This research assesses Ada's suitability in simple, non-embedded applications, specifically, numerical computation, simulation, and file processing. FORTRAN and Pascal programs in these applications were translated into Ada. Comparisons were made between the originals and the translations with regard to lines of source code, transportability, maintainability, readability, execution time, and any other finding relevant to the study. The study revealed that while further research is needed, Ada is a powerful programming language suitable for use in these non-embedded applications.

END

FILMED

1-86

DTIC